Edison
robotika pro každého

Pracovní listy EdPy

Studentské pracovní listy a listy s aktivitami

Jméno _____

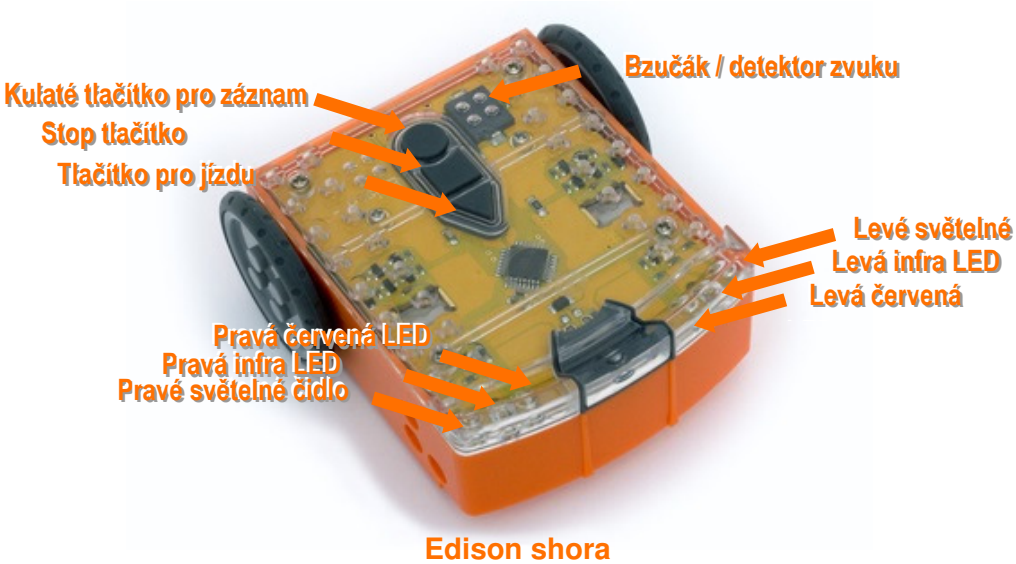
Obsah

Jméno _____

Lekce 1: Pracovní list 1.1 – Setkání s Edisonem

Edison je malý programovatelný robot. Edison k interakci se světem používá čidla (=senzory, snímače) a motory. Můžete také použít Edisona s kostičkami LEGO a stavět mnoho věcí.

Podívejte se na níže uvedené obrázky a seznamte se s Edisonovými čidly, tlačítky a přepínači.



Edison shora

Tlačítko pro jízdu (Play) — trojúhelníkové — Spustí program
Tlačítko stop — čtvercové — Stiskem zastaví program
Tlačítko pro záznam — kulaté — Jeden stisk je nahrání (download) programu z PC, tři stisky znamenají načtení čárového kódu jízdou robota přes kód

Vespod: Umístění spínače napájení a čidla pro sledování čáry.

Edisonův snímač pro sledování čáry se skládá ze dvou částí, z červené LED a světelného čidla.

Snímač pro sledování čáry si také může přečíst speciální čárové kódy, které aktivují přednastavené programy.

EdComm kablík se používá pro nahrávání vašich vlastních programů do Edisona. Zapojuje se do zdířky pro sluchátka v počítači nebo tabletu.

EdComm kablík k přenosu programu

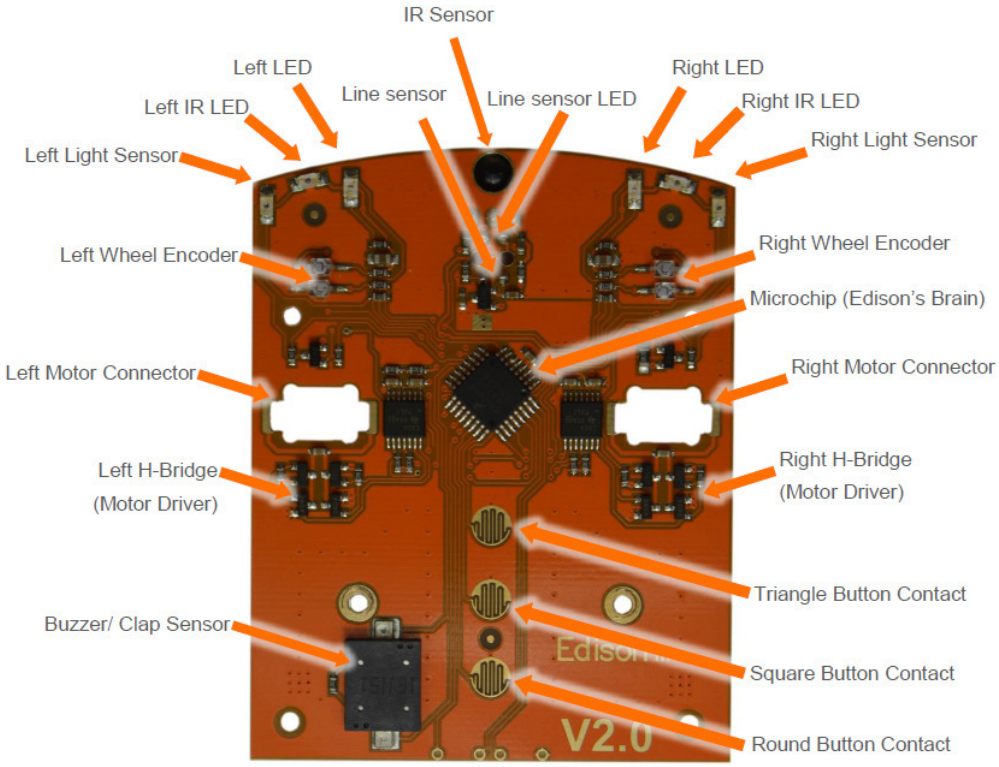
www.edpyapp.com

Jméno _____

Obrázek základní desky Edisona V2.0

Tento obrázek je platný pro verzi hardwaru Edisona V2.0

Oproti verzi V1 zde přibyly kodéry (snímače) polohy kol, tedy Edison dokáže měřit vzdálenost, kterou ujel (reálně měří, o jaký úhel se jedno nebo druhé kolo otočilo).



Buzzer / Clap Sensor	Bzučák / Čidlo tlesknutí
Left H-Bridge (Motor Driver)	Levý H-můstek (Pohon motoru)
Left Motor Connector	Konektor levého motoru
Left Wheel Encoder	Kodér levého kola
Left Light Sensor	Levé světelné čidlo
Left IR LED	Levá infračervená LED
Left LED	Levá LED
Line sensor	Čidlo čáry
IR sensor	Infračervené čidlo
Line sensor LED	LED čidla čáry
Right LED	Pravá LED
Right IR LED	Pravá infračervená LED
Right Light Sensor	Pravé světelné čidlo
Right Wheel Encoder	Kodér pravého kola
Microchip (Edison's Brain)	Mikročip/mikroprocesor (Edisonův mozek)
Right Motor Connector	Konektor pravého motoru
Right H-Bridge (Motor Driver)	Pravý H-můstek (Pohon motoru)
Triangle Button Contact	Kontakt trojúhelníkového tlačítka
Square Button Contact	Kontakt čtvercového tlačítka
Round Button Contact	Kontakt kulatého tlačítka

Jméno _____

Lekce 1: Pracovní list 1.2 – Programování čárovými kódy

Edison se dodává s předem připravenými programy, které se aktivují přejetím speciálních čárových kódů.

Čtení čárového kódu

1. Umístěte Edisona vpravo od čárového kódu nad šipku.
2. Stiskněte kulaté tlačítko záznam třikrát.
3. Edison pojede dopředu a naskenuje čárový kód.
4. Načtený program spusťte stisknutím tlačítka přehrávání (trojúhelník).

Čárový kód – Ovládání jízdy tlesknutím

Čárový kód – Vyhýbání se překážkám

Čárový kód – Sledování vodící čáry

Čárový kód – Sledování zdroje světla

Jste na řadě

Pro každý čárový kód popište, jakou akci robot udělá a proč to udělá.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

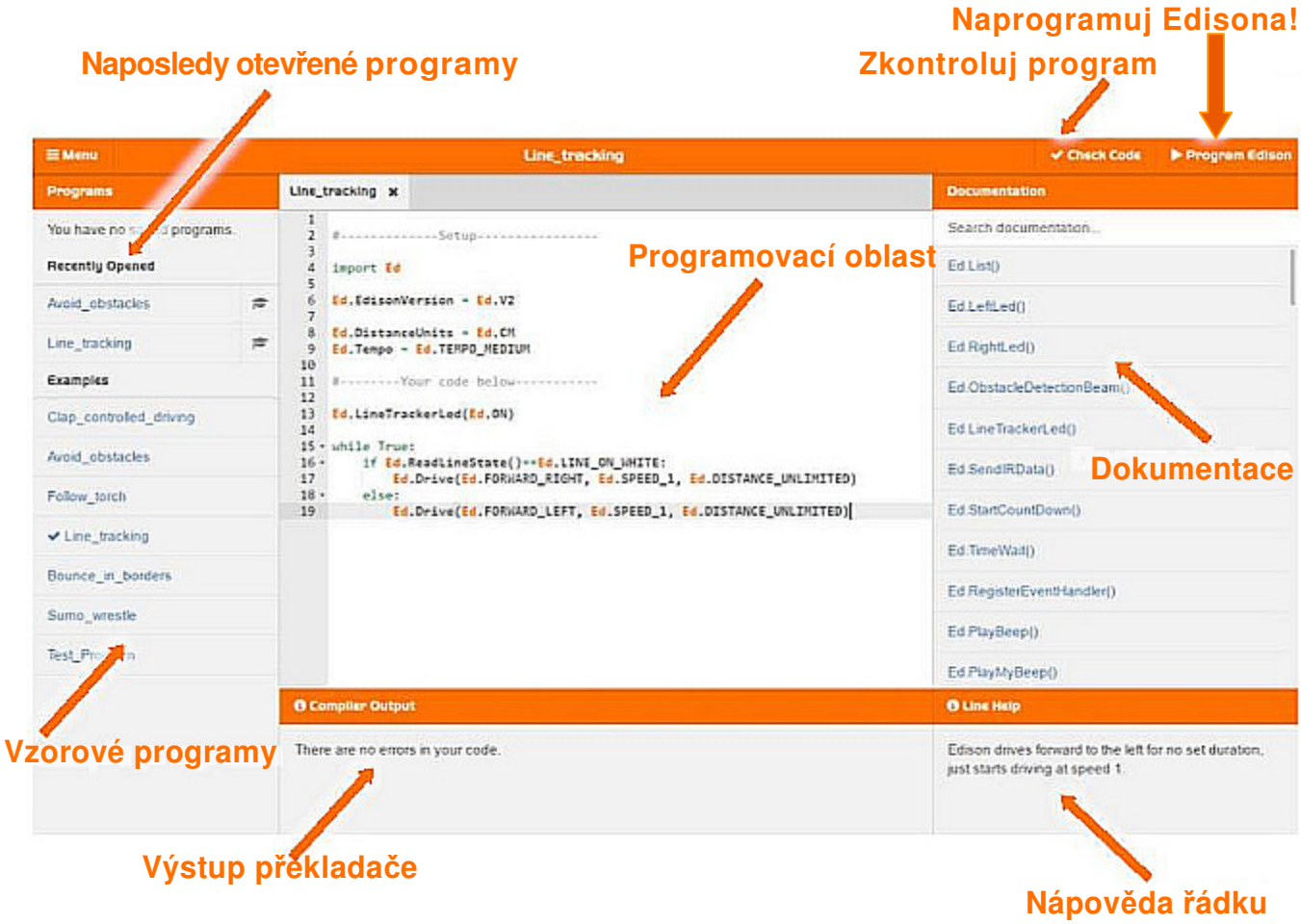
Jméno _____

Lekce 1: Pracovní list 1.3 – Setkání s aplikací EdPy

V této aktivitě se seznámíte s aplikací EdPy, což je software na počítači nebo tabletu, který budeme používat pro programování robota Edisona. V prohlížeči otevřete odkaz:

http://www.edpyapp.com/

Chcete-li se seznámit s aplikací EdPy a programováním, zkuste otevřít některé příklady programů. Vyhledáním v okně Documentation (Dokumentace) zjistěte, jak některé funkce fungují. V sekci dokumentace najdete vše, co potřebujete vědět o příkazech aplikace EdPy. Zkuste také pomocí funkce Line Help (Nápověda řádku) zjistit více o každé funkci.



Jste na řadě:

1. Jak byste měli změnit následující řádek, jestliže používáte Edisona verzi 1?

Ed.EdisonVersion = **Ed**.V2 _____

2. Kolik vstupních parametrů má každý z následujících příkazů?

Ed.PlayBeep() _____

Ed.TimeWait() _____

Ed.LeftLed() _____

Ed.DriveRightMotor() _____

www.edpyapp.com

Jméno _____

3. Nastavovací řádky programu můžete měnit tak, aby jako měrná jednotka v průběhu celého programu sloužily palce namísto centimetrů. Pokud jste to udělali, jak by měl být napsán příslušný nastavovací řádek?

_____

_____

4. Pokud se po kliknutí na tlačítko 'Zkontroluj program' najdou v programu chyby, ve kterém okně se tyto chyby objeví?

_____

_____

Jméno _____

Lekce 1: Pracovní list 1.4 – Stažení zkušebního programu

Otevřete program s názvem Test_Program z okna 'Examples' (Příklady) v EdPy.

Vzorový program ' Test_Program' vypadá takto:

Test_Program

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.TIME
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13
14
15 while True:
16     Ed.PlayBeep()
17     Ed.LeftLed(Ed.OFF)
18     Ed.RightLed(Ed.ON)
19     Ed.Drive(Ed.SPIN_RIGHT, 5, 350)
20     Ed.TimeWait(20, Ed.TIME_MILLISECONDS)
21     Ed.PlayBeep()
22     Ed.LeftLed(Ed.ON)
23     Ed.RightLed(Ed.OFF)
24     Ed.Drive(Ed.SPIN_LEFT, 5, 350)
25     Ed.TimeWait(20, Ed.TIME_MILLISECONDS)
26
```

Edison se podívá na každý řádek programu jeden po druhém a provádí, co řádek určuje.

Existují však některé řádky, které Edison vynechá (přeskočí).

Podívejte se na řádek 2, který začíná znakem " # " (hash, česky čteme 'heš'). Když řádek začíná tímto znakem, nazývá se "komentářový řádek". Edison bude ignorovat všechny znaky, které jsou na řádku po znaku " # " a přesune se na další řádek. Při programování používáme komentářový řádek pro dokumentování našeho programu (někdy **programu** říkáme **kód** (angl. 'code') a znamená to úplně to samé), abychom mohli sledovat a poznamenávat si své nápady a aby ostatní lidé program pochopili.

Stáhněte program do Edisona

Chcete-li program stáhnout do Edisona, připojte kabel EdComm do zásuvky pro sluchátka v počítači a zvyšte hlasitost. Zapojte druhý konec kabelu EdComm do Edisona podle obrázku.

Jméno _____

Chcete-li testovací program stáhnout (někdy se může použít české výstižnější slovo **'nahrát'**) do Edisona, postupujte takto:

1. Zapněte Edisona a stiskněte jednou tlačítko záznamu na Edisonovi (kulaté)
2. Připojte Edisona k počítači pomocí kabelu EdComm a ověřte si, že je hlasitost nastavená na maximum
3. Stiskněte tlačítko **'Program Edison'**, které česky znamená **Naprogramuj Edisona!**, toto tlačítko se nachází v pravém horním rohu aplikace EdPy. (Pro stručnost výkladu už v dalším textu nebudeme upozorňovat na přesný **český překlad** názvu tohoto tlačítka!)
4. Postupujte podle pokynů v rozbalovacím okně a stiskněte tlačítko 'Program Edison'

Po skončení stahování programu odpojte kabel EdComm. Jedním stisknutím tlačítka přehrávání (trojúhelník) program spustíte.

Jste na řadě:

1. Co udělal robot při stisknutí tlačítka přehrávání?

_____

_____

_____

_____

2. Podívejte se na příkazy Pythonu v programu a přemýšlejte o tom, co Edison dělal, když jste program přehrávali (spustili). Popište, v jakém vzájemném vztahu tyto příkazy jsou.

_____

_____

_____

_____

3. Vysvětlete, jak program dostane z počítače do robota.

_____

_____

_____

_____

Jméno _____

Lekce 2: Pracovní list 2.1 – Jízda robota vpřed

V této aktivitě potřebujete napsat program, který by jel s Edisonem vpřed.

Podívejte se na následující program:

```
1
2  #-------------Setup---------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below----------
12
13 Ed.Drive(Ed.FORWARD,Ed.SPEED_5,8)
14
```

Krok 1: Začněte psát výše uvedený program do aplikace EdPy zadáním příkazu "Ed" na řádku 13.

Jakmile začnete psát 'Ed' na řádku 13, vyskočí okénko se seznamem (odborně se tomuto způsobu nápovědy příkazů někdy složitě říká 'dokončení příkazového řádku', jindy 'pole se seznamem' a někdy taky daleko lidštěji '**našeptávání textu**', a my tomu budeme říkat zkráceně **'našeptávač'**). Nebojte se, slovo '**našeptávač**' se už v češtině docela zabydlelo, je krátké a všem srozumitelné. Toto pole/vyskakovací okénko/našeptávač zobrazuje seznam možných příkazů, které si lze v danou chvíli (a zejména v daném kontextu) vybrat. Takový výběr příkazů (a jindy třeba výběr parametrů) velice usnadňuje a zrychluje programování. Příkazy téměř 'nelze' napsat s překlepem. Našeptávač vás ale neochrání před výběrem příkazu úplně nesprávného.

Krok 2: Do řádku 13 zadejte 'Ed.Drive' a v našeptávači vyberte funkci 'Ed.Drive()'.

Ed.Drive() je (předem připravená) funkce v Pythonu, která je importována z knihovního modulu Edisona, modul se nazývá 'Ed' a import je vyvolán nastavovacím řádkem.

Funkce je část programu, která vykonává určitou roli nebo úlohu, v závislosti na zadaných vstupních parametrech. Všechny funkce, které jsou importovány z knihovny 'Ed', musí začínat na **'Ed.'** To informuje program, do které knihovny má jít, aby tuto funkci našel.

Krok 3: Vyplňte vstupní parametry.

Pokud má nějaká funkce vstupní parametry, musíte zadat hodnotu pro každý z nich.

Funkce **Drive()** má tři vstupní parametry:

- **směr** – směr, kterým Edison pojede
- **rychlost** – rychlost, se kterou Edison pojede

- **vzdálenost** – počet jednotek vzdálenosti, které Edison ujede

Různé vstupní parametry mají různé hodnoty. Např. rychlost se zadává **Ed**.SPEED_ a k tomu připíšeme číslo od 1 do 10 (10 je maximum). V obrázku vidíme **Ed**.SPEED_5

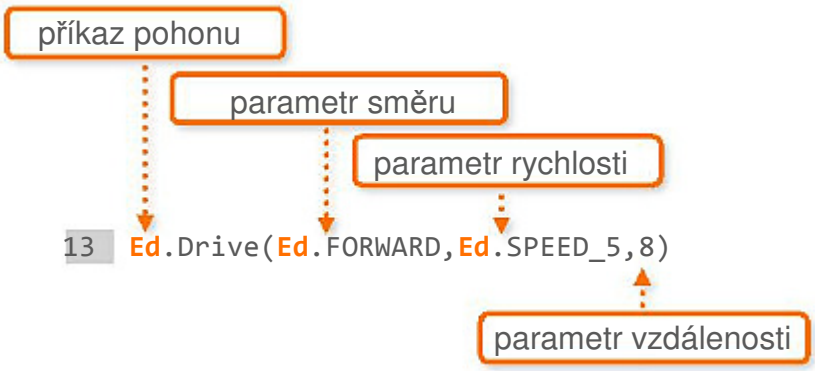
Typ jednotek vzdálenosti *[čili fyzikální jednotky použité pro vzdálenost]* je řízen konstantou, která se vepíše (přiřadí) do hodnoty proměnné **Ed**.DistanceUnits v nastavovací části programu. K dispozici jsou tři jednotky vzdálenosti, které lze použít:

- centimetry, napsané jako **Ed**.CM
- palce, napsané jako **Ed**.INCH
- čas, napsaný jako **Ed**.TIME

Poznámka: Pokud máte robot Edison ve verzi V1, smíte použít jenom čas, takže se ujistěte, že **Ed**.DistanceUnits = **Ed**.TIME. Jednotka časové vzdálenosti jsou milisekundy, to znamená, že k jízdě po dobu 2 sekund musíte nastavit vstupní parametr na hodnotu 2000 (protože 2000 milisekund = 2 sekundy).

Krok 4: Zkontrolujte svůj program.

Podívejme se blíže na funkci pohonu a parametry v tomto programu.



Jakmile napíšete celý program, klikněte na tlačítko 'Zkontroluj program' a podívejte se na okno 'Výstup překladače', abyste se ujistili, že jste při psaní neudělali žádné chyby.

Chyby typu překlep se nazývají syntaktické chyby. Pokud zadáte slovo, které není součástí jazyka programu EdPy (čili odborně řečeno není součástí tzv. 'syntaxe' jazyka EdPy), pak překladač EdPy nemůže pochopit, co má dělat, a ohlásí se chyba syntaxe.

Pokud jsou ve vašem programu chyby, opravte je, aby program odpovídal příkladu.

Krok 5: Stáhněte a otestujte svůj program.

Po kontrole, že váš program neobsahuje žádné chyby, jej stáhněte (nahrajte) do Edisona.

1. Zapněte Edisona a stiskněte jednou tlačítko záznamu na Edisonovi (kulaté)
2. Připojte Edisona k počítači pomocí kabelu EdComm a ověřte si, že je hlasitost nastavená na maximum
3. Stiskněte tlačítko "Program Edison" v pravém horním rohu aplikace EdPy
4. Postupujte podle pokynů v rozbalovacím okně a stiskněte tlačítko 'Program Edison'

Jméno _____

Jakmile se program stáhne, odpojte kabel EdComm. Použijte list aktivity 2.1 nebo barevnou pásku pro vyznačení čar 'start' a 'cíl' na stole nebo podlaze jako zkušební oblast pro program. Jedním stisknutím tlačítka přehrávání (trojúhelník) spusťte program a sledujte, co se stane.

Krok 6: Experimentujte se svým programem.

Změřte vzdálenost mezi počáteční čárou a cílovou čarou. Pokuste se upravit svůj program tak, aby robot Edison dojel těsně před cílovou čáru. Pokuste se zjistit, co funguje.

Jste na řadě:

1. Jakou konstantu jste nastavili jako "Ed.DistanceUnits" v nastavovací části programu?

_____

2. Jaké číslo jste museli zadat jako vstupní parametr pro vzdálenost, aby Edison přejel od startovací k cílové čáře?

_____

3. Experimentujte s jízdou Edisona při různých rychlostech. Co dělá robot, když ho poháníte rychlostí 10? Všimli jste si nějakých změn v přesnosti Edisona při jízdě rychlostí 10?

_____

_____

_____

Velká poznámka překladatele k ukázce programu.

1. Předpona `Ed.` vyjadřuje, že se slova `Tempo` a `TEMPO_MEDIUM` budou hledat ve speciálním souboru (tzv. knihovně) s názvem `Ed`. Překladač Pythonu pomocí nastavovacího řádku `import Ed`. v úseku `Setup` vyzveme, aby si tento soubor našel a pracoval s ním.

Pozor, úsek `#----------Setup------` i úsek `#---------Your code below-----` nejsou 'oficiálními' strukturami jazyka Python, ale jsou to jen naše vlastní komentářové poznámky pro **naši** lepší orientaci v programu.

2. Všimněte si rozdílu v tomto zápisu: `Ed.Tempo` oproti `Ed.TEMPO_MEDIUM`

   a) `Tempo` obsahuje velká i malá písmena – je to název (jméno) **proměnné**

   b) `TEMPO_MEDIUM` obsahuje pouze velká písmena – je to název (jméno) **konstanty**

Je to zase **naše** dohoda, jak budeme názvy zapisovat – zlepšíme tak čitelnost programu, myslí se tím hlavně čitelnost pro lidi – programátory. Počítači/překladači by to bylo jedno.

3. Povolené znaky ve jménech proměnných, **konstant**, funkcí atd. jsou **písmena**, **číslice** a "_". Jméno nesmí začínat číslicí. Pozor, velká a malá písmena se **rozlišují**!

Jméno _____

Lekce 2: Pracovní list 2.2 – Jízda robota zpět

V této aktivitě potřebujete napsat program, který by jel s Edisonem zpět.

Kdykoli píšete program pro Edisona v EdPy, vždy postupujte podle stejných základních kroků:

- **Krok 1:** Zkontrolujte, zda nastavovací řádky používají požadované konstanty.
- **Krok 2:** Pište program, zvolte funkce, které chcete použít a vyplňte jejich vstupní parametry požadovanými hodnotami.
- **Krok 3:** Pomocí tlačítka 'Zkontroluj program' jej zkontrolujte.
- **Krok 4:** Stáhněte a otestujte program pomocí robota Edisona.

*Pamatujte, že pokud máte robota ve verzi Edison V1, zajistěte, aby **Ed**.DistanceUnits = **Ed**.TIME. Parametr vzdálenosti pro **Ed**.TIME je v milisekundách.*

Jste na řadě:

Úkol 1: Jeďte zpět – couvejte

Napište následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 Ed.Drive(Ed.BACKWARD,Ed.SPEED_5,8)
14
```

Použijte list aktivity 2.1 nebo barevnou pásku pro vyznačení čar 'start' a 'cíl' na stole nebo podlaze jako zkušební oblast pro program. Experimentujte se svým programem, zda dokážete s Edisonem couvat od startovní k cílové čáře.

Úkol 2: Použijte konstantu '**Ed**.DISTANCE_UNLIMITED'

Existuje mnoho způsobů, jak programovat Edisona, aby jezdil dopředu a dozadu. Dalším způsobem je použití parametru '**Ed**.DISTANCE_UNLIMITED' pro parametr vzdálenosti. Tato konstanta zapne oba motory Edisona.

Jméno _____

Na rozdíl od číselné hodnoty parametru vzdálenosti, konstanta Ed.DISTANCE_UNLIMITED neurčuje přesnou hodnotu, po které se motory zastaví. Místo toho zapne motory a poté se vykonávání programu přesune na další řádek kódu. Zastavení motorů se bude muset udělat později v kódu.

Použití konstanty **Ed**.DISTANCE_UNLIMITED může být užitečné, pokud chcete napsat program, kde se motory zastaví, jenom když nastane jiná událost, např. když je zjištěna překážka.

Podívejte se na následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 Ed.Drive(Ed.BACKWARD, Ed.Ed.SPEED_6, Ed.DISTANCE_UNLIMITED)
14 Ed.TimeWait(200, Ed.TIME_MILLISECONDS)
15 Ed.Drive(Ed.STOP, Ed.SPEED_10, 0)
16
```

Tento program zapne motory Edisona pro pohyb zpět a pak čeká 200 milisekund a motory vypne.

Napište program pomocí parametru **Ed**.DISTANCE_UNLIMITED pro jízdu zpět. Použijte list činností 2.1 nebo barevnou pásku, abyste vyznačili čáry 'start' a 'cíl' na stole nebo na podlaze jako zkušební oblast pro program. Experimentujte s programem, abyste zjistili, zda můžete přimět Edisona, aby couval od cílové na startovní čáru.

1. Nastavte rychlost v programu na **Ed**.SPEED_6. Kolik milisekund je třeba zadat do funkce TimeWait(), aby Edison couval zpět od cílové na startovní čáru?

_____

2. Experimentujte s různými rychlostmi a časovými parametry TimeWait(). Jaké jsou nejrychlejší a nejpomalejší časy, kdy ještě může Edison couvat od cílové na startovní čáru?

Nejrychlejší: _____ Nejpomalejší: _____

Jméno _____

Lekce 2: Pracovní list 2.3 – Vpřed, pak zpět

V této aktivitě potřebujete napsat program, který by jel s Edisonem vpřed a pak zpět.

Kdykoli píšete program pro Edisona v EdPy, vždy postupujte podle čtyřech základních kroků:

- **Krok 1:** Zkontrolujte, zda nastavovací řádky používají požadované konstanty.
- **Krok 2:** Pište program, zvolte funkce, které chcete použít a vyplňte jejich vstupní parametry požadovanými hodnotami.
- **Krok 3:** Pomocí tlačítka 'Zkontroluj program' jej zkontrolujte.
- **Krok 4:** Stáhněte a otestujte program pomocí robota Edisona.

*Pamatujte, že pokud máte robota ve verzi Edison V1, zajistěte, aby **Ed**.DistanceUnits = **Ed**.TIME. Parametr vzdálenosti pro **Ed**.TIME je v milisekundách.*

Jste na řadě:

Úkol 1: Napište následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 Ed.Drive(Ed.FORWARD,Ed.SPEED_5,8)
14 Ed.Drive(Ed.BACKWARD,Ed.SPEED_5,8)
15
```

Použijte list činností 2.1 nebo barevnou pásku, abyste vyznačili čáry 'start' a 'cíl' na stole nebo na podlaze jako zkušební oblast pro program.

1. Jaké jsou správné hodnoty vstupních parametrů vzdálenosti tak, aby se Edison jel dopředu a pak zpět mezi startovní a cílovou čarou?

Vpřed _____ Zpět _____

Úkol 2: Napište nový program, díky němuž se Edison pojede dopředu, pak zpět mezi startovní a cílovou čárou. Tentokrát použijte parametry **Ed**.DISTANCE_UNLIMITED a funkce **Ed**.TimeWait().Použijte list činností 2.1 nebo barevnou pásku, abyste vyznačili čáry 'start' a 'cíl' na stole nebo na podlaze jako zkušební oblast pro program.

Jméno _____

2. Jak vypadá váš nový program? Napište program sem.

_____

_____

_____

_____

_____

_____

_____

_____

_____

Jméno _____

Lekce 2: Pracovní list 2.4 – Výrazy v jazyce Python

V této aktivitě se dozvíte o důležitém prvku kódu (zápisu programu), který používáme při programování v jazyce Python – jsou to **výrazy**.

Co to jsou výrazy?

Výraz je otázka, která může být rozhodnuta buď jako 'pravda' nebo 'nepravda'.

*Překladatel vám nebude tajit, že odedávna a ve většině programovacích jazyků jde o anglická slova **true** a **false**. A ještě jedn o důležité upozornění, v Pythonu pro hodnoty **true** a **false** existují dvě samostatná klíčová slova, tato slova se píšou s velkým počátečním písmenem takto:* `True` *a* `False` *a aplikace EdPy je podbarvuje zeleně. A protože v počítači je vše vyjádřeno jako číslo, tak si ještě řekneme, že v Pythonu je* `True=1` *a* `False=0`*.*

Příklady výrazů: 'Je A menší než B?' Nebo 'Je A stejné jako B?"

V programu jsou výrazy psány pomocí matematických zápisů namísto slov.

V nastavovacích řádcích jste viděli zápis (notaci) s použitím rovnítka **=**.
Např. jsme použili toto: **Ed**.DistanceUnits = **Ed**.CM.
Když zapíšeme 'A = B', znamená to 'nastav A na stejnou hodnotu jako má B'.

Pozn. překl.: Bohužel tento ***příklad s rovnítkem*** není ve výkladu vhodně načasován a je tu značně matoucí. Zde v tomto příkladu nejde o ty "otázky/dotazy" na veličiny z úvodu kapitoly (tedy **výrazy**), ale jde o ***jinou část jazyka Python*** zvanou **příkazy**, které říkají, že se má něco někam zkopírovat/zapamatovat/nastavit. Plyne z toho zatím jediný závěr: Python naštěstí rozlišuje jednoduché rovnítko '=' a dvojité rovnítko '==' a každé z nich znamená něco úplně jiného! To první je **příkaz 'A = B'** a to druhé je **výraz 'A == B'**. Existuje ale také mnoho programovacích jazyků, kde dvojité rovnítko prostě není a musíme pak sami poznat, jakou roli rovnítko v daném místě zrovna hraje.

Výrazy používají různé zápisy (notace). Toto jsou některé ze základních zápisů (notací) ve výrazech. Zápisy výrazů, které můžeme použít v Pythonu, jsou zde:

Výraz	Význam výrazu
A == B	Je A rovno B?
A != B	Je A různé od B?
A > B	Je A větší než B?
A >= B	Je A větší nebo rovno B?
A < B	Je A menší než B?
A <= B	Je A menší nebo rovno B?

Výrazy porovnávají mezi sebou levou stranu a pravou stranu zápisu (notace).

V seznamu výše uvedených výrazů můžete nahradit 'A' a 'B' libovolnou **hodnotou** nebo také **funkcí**, která **vrací hodnotu**. [Pozn. př. **Vrátit hodnotu** je v programování zásadní a důležitý pojem a my si zde zatím jen zapamatujeme, že toto existuje. Podrobný výklad se sem nevejde a budeme se tomu věnovat později, nebo pokročilí mohou zapojit i samostudium.]

Jméno _____

S těmito **hodnotami** můžeme také **matematicky počítat**!

Např.: '(A + 2)> B' znamená dotaz 'Je (A plus 2) větší než B?'

V programu se pracuje s výrazy v určitém pořadí. Když výraz obsahuje matematické zápisy ('matematiku') nebo volá funkci, výraz vyřeší nejprve matematiku nebo funkce. Potom porovná levou stranu výrazu s pravou stranou a rozhodne buď 'true' nebo 'false'.

Jak se výrazy používají v Pythonu?

Výrazy se používají společně s dalšími prvky kódu, jako jsou ***příkazy pro smyčky* 'while'** (častěji se smyčkám říká ***cykly***, pozn. př. a česká wikipedie) a společně s ***podmíněnými příkazy* 'if'**, které mění způsob toku programu, neboli dovolí jeho rozvětvení v závislosti na nějaké podmínce (podmínka je nám už známý výraz). Tyto cykly a podmíněné příkazy umožňují, aby se tok programu pohyboval jiným způsobem, než již dříve zmíněným nejjednodušším postupem po řádcích takto: řádek 1 → řádek 2 → řádek 3 atd.

Jste na řadě:

Zkoušejte řešit výrazy. Nejprve si napište, co znamená každý výraz, a pak rozhodněte, zda je výsledek výrazu true nebo false.

Když bude A = 2 a B = 4, co znamená každý z následujících výrazů a jaký je jeho výsledek (true nebo false)?

1. **(A*2) == B**

Význam: _____

Výsledek výrazu: _____

2. **A >= B**

Význam: _____

Výsledek výrazu: _____

3. **(A+A) != B**

Význam: _____

Výsledek výrazu: _____

4. **(A-1) < (B-3)**

Význam: _____

Výsledek výrazu: _____

Jméno _____

Lekce 2: Pracovní list 2.5 – Jízda aktivovaná klávesnicí

V této aktvitě máme napsat program, který rozjede Edisona dopředu pouze tehdy, když je stisknuto buď kulaté nebo trojúhelníkové tlačítko. K tomu použijeme **výrazy** a cyklus **while**.

Podívejte se na tento program:

```
1
2  #------------Setup---------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 Ed.ReadKeypad()
13 while Ed.ReadKeypad() == Ed.KEYPAD_NONE:
14     pass
15 Ed.Drive(Ed.FORWARD, Ed.SPEED_6,8)
16
```

Program používá cyklus **while** s výrazem.

V Pythonu se v cyklu **while** opakuje jeden příkaz (nebo skupinu příkazů) po dobu, kdy daná podmínka má hodnotu **true**. Stav podmínky (výrazu) se testuje pokaždé **před** provedením cyklu. [Pozn. př.: V Pythonu existují i jiné cykly s podmínkou.]

Po celou dobu, kdy je výraz OPAKOVANĚ vyhodnocován jako **true**, program opakuje příkazy v cyklu. Když se výraz za 'while' vyhodnotí jako **false**, cyklus se ukončí (a už se neprovede ani jednou) a program se přesune na svůj další řádek za cyklem.

Pozn. př.: Musím opět výslovně zdůraznit, že výraz za slovem **while** je vyhodnocen pokaždé znova před každým pokusem vykonat další průchod cyklem. Změnou proměnných ve výrazu za **while** můžeme tedy zevnitř cyklu ovlivnit, kdy cyklus skončí.

O příkazech uvnitř cyklu také někdy hovoříme jako o 'těle cyklu'.

*Pozn. př.: Hned od počátku si všimněte dvojtečky za každou podmínkou, je **povinná**!*

Použití odsazení

Python používá pro seskupení některých příkazů dohromady ' odsazení'.

V Pythonu jsou všechny příkazy odsazené stejným počtem mezer považovány za jeden blok programu. [V předchozím odstavci byl takový blok programu tělo cyklu.]

Podívejte se na řádek 14 programu. Protože slovo **pass** je odsazeno, tak to znamená, že je uvnitř cyklu **while**, který začal na řádku 13. Řádek 15 v programu není odsazen, a proto řádek 15 UŽ NENÍ uvnitř cyklu **while** ze řádku 13.

Pozn. př.: Existují jiné programovací jazyky, kde je blok programu vyznačen jinak, například slovy *begin* a *end* nebo speciálními značkami – třeba závorkami { a }.

Jméno _____

Funkce a konstanty v tomto programu

Ed.ReadKeyPad() – tato funkce čte Edisonův stav klávesnice. Jinými slovy určuje, zda na Edisonovi bylo stisknuto jedno z jeho tlačítek nebo ne.

Funkce **Ed**.ReadKeyPad() vrátí hodnotu vyjadřující, které tlačítko bylo stisknuto: **Ed**.KEYPAD_NONE, **Ed**.KEYPAD_TRIANGLE nebo **Ed**.KEYPAD_ROUND.

Tato funkce nefunguje pro čtvercové tlačítko. Je to proto, že čtvercové tlačítko je určeno pouze k zastavení programu. Čtvercové tlačítko vždy zastaví běžící program.

Zvláštní poznámka: Použití funkce 'čtení' uvnitř smyčky

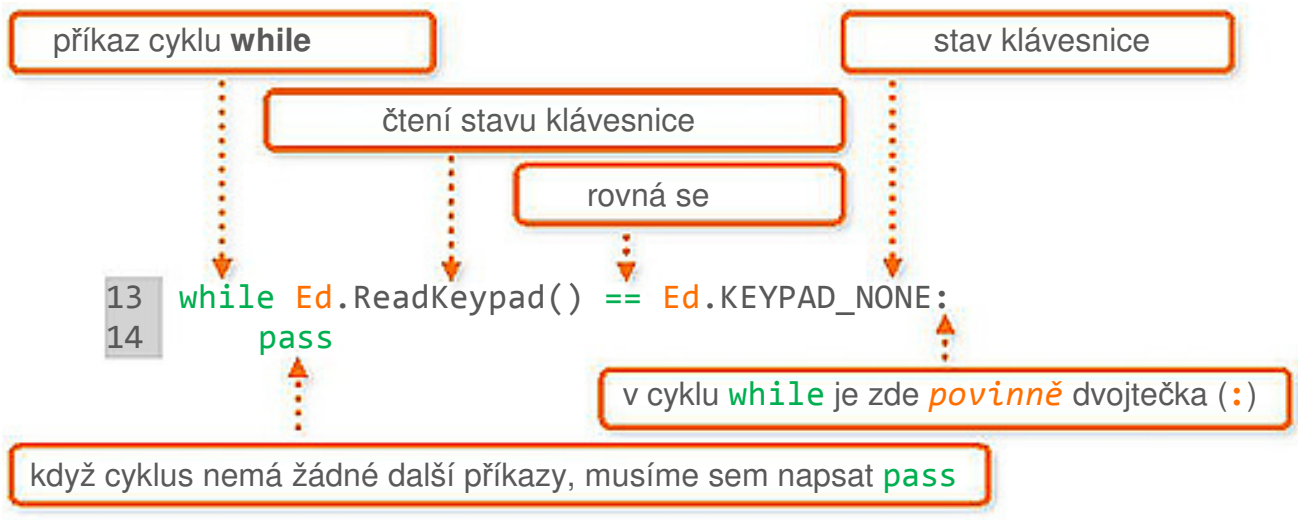
Některé typy dat se dočasně ukládají do paměti Edisona. Takto může funkce **Ed**.ReadKeyPad() načíst stisknutí tlačítka, ke ktrému došlo ještě před zavoláním funkce čtení ve vašem programu.

V tomto programu se chceme ujistit, že **Ed**.ReadKeyPad() v cyklu **while** bude správně čekat, dokud nebude skutečně nově stisknuto tlačítko, a nebrat v úvahu žádné stisknutí tlačítek, které by se případně stalo před cyklem **while**. To je důvod, proč jsme **Ed**.ReadKeyPad() vložili do řádku nad cyklem **while** (na řádek 12). Tímto vymažete z paměti všechna předcházející stisknutí kláves před cyklem.

Takto byste měli postupovat vždy při použití funkce čtení uvnitř cyklu.

Jste na řadě:

Napiště program. Ověřte si, že jste při psaní cyklu **while** napsali dvojtečku a správně odsazovali.



Stáhněte program a stisknutím tlačítka trojúhelníku program spusťte. Počkejte chvíli a zkuste stisknout buď trojúhelníkové nebo kulaté tlačítko.

Jméno _____

1. Co Edison udělal při stisknutí trojúhelníkového nebo kruhového tlačítka několik sekund po spuštění programu?

_____

_____

2. Spusťte program znovu a zkuste stisknout čtvercové tlačítko namísto kruhového nebo trojúhelníkového tlačítka. Co se stalo? Proč se to stalo? Vysvětlete.

_____

_____

_____

_____

3. Nyní zkuste přidání dalšího kódu na konec programu. Nový kód, který napíšete, by měl po stisknutí kulatého nebo trojúhelníkového tlačítka Edisona rozjet zpět. Jinými slovy, program by měl Edisonovi říci, aby se po prvním stisknutí tlačítka rozjel dopředu, a pak, když se znovu stiskne tlačítko, jel dozadu (couval). Nezapomeňte zahrnout dvojtečku a odsadit kód uvnitř cyklu while. Jak vypadá váš nový program? Sem si jej napište:

_____

_____

_____

_____

_____

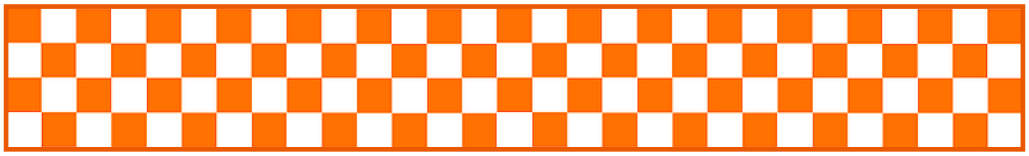
_____

_____

_____

Jméno _____

Lekce 2: List aktivity 2.1



CÍL

START

Jméno _____

Lekce 3: Pracovní list 3.1 – Zatočte doprava

V této aktivitě musíte napsat program, který otočí Edisona doprava.

Podívejte se na tento program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 degreesToTurn = 90
13 Ed.Drive(Ed.SPIN_RIGHT,Ed.SPEED_6,degreesToTurn)
14
```

Podívejte se na řádek 13 programu. Nezapomeňte, že funkce **Ed**.Drive má tři vstupní parametry: směr, rychlost a vzdálenost. V tomto programu není parametr vzdálenosti dán číslem, ale je zde napsáno **degreesToTurn**.

Toto **degreesToTurn** je proměnná!

V Pythonu jsou pro proměnné vyhrazena místa paměti, kde se ukládají jejich hodnoty. To znamená, že při vytváření proměnné se v paměti programu vyhradí nějaké místo.

Proměnná představuje hodnotu, která se nastaví někde ve vašem programu.

Podívejte se na řádek 12 programu. Zde je místo, kde je nastavena hodnota pro proměnnou **degreesToTurn**. Tento postup se nazývá přiřazení hodnoty proměnné. V Pythonu se pro přiřazení hodnoty k proměnné používá rovnítko '**=**'.

Proměnné můžete použít k ukládání hodnot, které se používají na několika místech v celém programu. To může být velmi užitečné, zejména pokud se hodnota proměnné mění. *[Pozn. př.: Proměnná se jmenuje proměnná **právě proto**, že se MŮŽE měnit. Naopak konstanty se měnit nemohou.]* Když použijete proměnnou, tak stačí provést změnu pouze v jednom řádku kódu, abyste nastavili hodnotu všude tam, kde je tato proměnná v programu použita.

Jste na řadě:

Napište program, poté stáhněte a spusťte program v Edisonovi. Použijte list aktivity 3.1 nebo barevné pásky pro vyznačení úhlů 'začátek' a 'konec' na stole nebo na podlaze jako zkušební oblast pro váš program.

1. Popište, co robot dělá a proč to dělá, když je spuštěn program.

_____

_____

_____

Jméno _____

Přidejte nový řádek na konec programu (za řádek 13) a přidejte do programu následující kód:

```
Ed.Drive(Ed.SPIN_LEFT,Ed.SPEED_6,degreesToTurn)
```

Stáhněte a spusťte program s Edisonem ve zkušební oblasti.

2. Popište, co robot dělá a proč to dělá při spuštění aktualizovaného programu.

_____

_____

_____

Nyní upravte svůj program tak, aby se Edison nejprve otočil o 180 stupňů doprava a pak o 180 stupňů doleva.

Rada: Nezapomeňte, že můžete změnit hodnotu proměnné **degreesToTurn**.

Stáhněte a spusťte program s Edisonem ve zkušební oblasti, abyste zjistili, zda byla vaše změna úspěšná.

3. Jaký řádek/řádky jste ve svém programu změnili? Zapište sem aktualizované řádky.

_____

_____

_____

Názvy proměnných se musí řídit určitými pravidly Pythonu. Např. nejsou povoleny žádné zvláštní znaky, např. # nebo $. Zkuste změnit název proměnné **degreesToTurn**. Pokuste se experimentovat s různými možnými názvy a pomocí tlačítka 'Zkontroluj program' zjistěte, která jména jsou povolena a která povolena nejsou.

4. Uveďte dva příklady nepřípustných názvů proměnných, které jste objevili.

_____

_____

Jméno _____

Lekce 3: Pracovní list 3.2 – Otočte se doleva o 180°

V této aktivitě je třeba napsat dva různé programy, aby se váš Edison robot otočil doleva přesně o 180°.

Jste na řadě:

Úkol 1: Otočte se doleva přesně o 180°.

Napište program, díky němuž se Edison otočí doleva přesně o 180°.

Rada: Zkuste použít jako výchozí program ten program, který jste použili v listu 3.1.

Stáhněte svůj program a vyzkoušejte jej pomocí listu aktivity 3.1 nebo barevné pásky pro vyznačení úhlů 'začátek' a 'konec' na stole nebo na podlaze jako zkušební oblast pro váš program. Nezapomeňte se svým programem experimentovat. Pokud se Edison neotočí přesně o 180°, zkuste upravit vstupní parametry a znovu program testovat.

1. Jaké jsou vstupní parametry, které jste použili k tomu, aby se robot otočil přesně o 180°? Pokud jste použili proměnnou, uveďte jakou hodnotu jste pro danou proměnnou přiřadili.

_____

_____

_____

Úkol 2: Otočte se přesně o 180° pomocí příkazu **Ed**.DriveRightMotor().

EdPy má příkaz nazvaný **Ed**.DriveRightMotor(), který pohybuje pouze Edisonovým pravým motorem. Pokud se pohybuje pouze pravý motor, jak se Edison otočí? Držte Edisona v ruce a napodobte, co se stane, pokud se pohybuje pouze pravý motor.

Vyhledejte příkaz **Ed**.DriveRightMotor() v okně Dokumentace aplikace EdPy, abyste zjistili, jak tato funkce funguje.

Poté napište program, který robota otočí doleva o 180° pomocí funkce **Ed**.DriveRightMotor().

2. Jaké jsou vstupní parametry, které jste použili k tomu, aby se robot otočil doleva přesně o 180° pomocí příkazu **Ed**.DriveRightMotor()?

_____

_____

_____

Jméno _____

Lekce 3: Pracovní list 3.3 – Otočení doprava a pak doleva

V této aktivitě je třeba napsat program tak, aby se Edison otočil po stisknutí trojúhelníkového tlačítka.

Napište následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 degreesToTurn = 90
13 Ed.ReadKeypad()
14 while Ed.ReadKeypad() != Ed.KEYPAD_TRIANGLE:
15     pass
16 Ed.Drive(Ed.SPIN_RIGHT,Ed.SPEED_6,degreesToTurn)
17
```

Stáhněte svůj program a vyzkoušejte jej pomocí listu aktivity 3.1 nebo barevné pásky pro vyznačení úhlů 'začátek' a 'konec' na stole nebo na podlaze jako zkušební oblast pro váš program.

Jste na řadě:

Napište program, aby se robot otočil doprava přesně o 90° po jednom stisku trojúhelníkového tlačítka, a poté po druhém stisknutí trojúhelníkového tlačítka se otočil doleva přesně o 270°.

Nezapomeňte umístit **Ed**.ReadKeyPad() do řádku nad každým cyklem **while** pro vymazání všech předcházejících tlačítek před cyklem.

1. Jak vypadá váš program? Napište ho sem:

_____

_____

_____

_____

_____

_____

Jméno _____

Lekce 3: Pracovní list 3.4 – Minibludiště

V této aktivitě máte napsat program, který umožní Edisonovi úspěšně projít bludištěm.

Jste na řadě:

Napište program tak, aby váš Edison projel minibludištěm na listu aktivity 3.2, když stisknete tlačítko přehrávání (trojúhelník).

Chcete-li úspěšně dokončit bludiště, tak:

- Edison musí začít za čárou START
- Edison musí zastaví po překročení čáry CÍL a
- Edison musí zůstat uvnitř hranice bludiště.

Použijte znalosti programování robotů, které jste dosud získali, abyste napsali program, který využívá více funkcí, aby Edison projel zatáčkami bludiště.

Rady: **Ed**.Drive() **Ed**.SPIN_RIGHT **Ed**.FORWARD **Ed**.SPIN_LEFT

1. Popište sekvenci pohybu robota, aby dokončil bludiště.

_____

_____

_____

2. Co jste při psaní tohoto programu považovali za obtížné?

_____

_____

_____

_____

_____

Výzva 1: Závod!

Kdo projede bludištěm nejrychleji, bez podvádění?

Pamatujte si, že váš robot musí startovat za startovní čárou, zastavit se po cílové čáře a nesmí přejet žádné hraniční čáry, když chce vyhrát.

Jméno _____

3. Kdo závodil? Kdo závod vyhrál?

Soupeř: _____

Vítěz: _____

4. Jaký byl čas vítězného robota?

_____

Výzva 2: Navrhněte své vlastní bludiště

Navrhněte své vlastní, náročnější bludiště s několika dalšími zatáčkami, aby mohl Edison hledat správný směr. Napište program, aby Edison bludiště úspěšně dokončil. Nebo si vyměňte bludiště s partnerem a napište program, který úspěšně dokončí jeho bludiště.

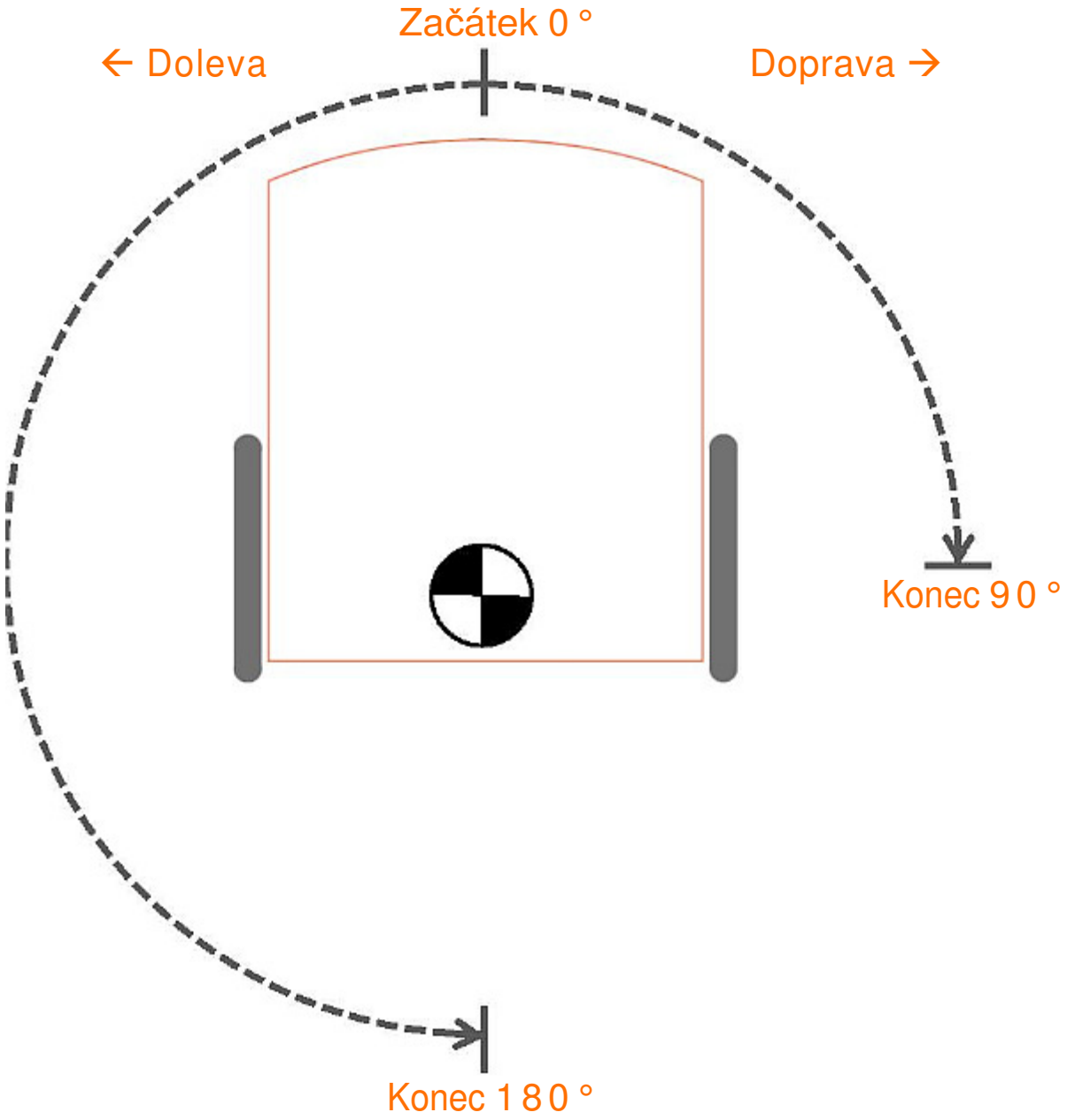
Pamatujte si, že váš robot musí startovat za startovní čárou, zastavit se po cílové čáře a nesmí přejet žádné hraniční čáry, když chce vyhrát.

5. Do tohoto okénka nakreslete malou verzi bludiště, které jste projeli.

Jméno _____

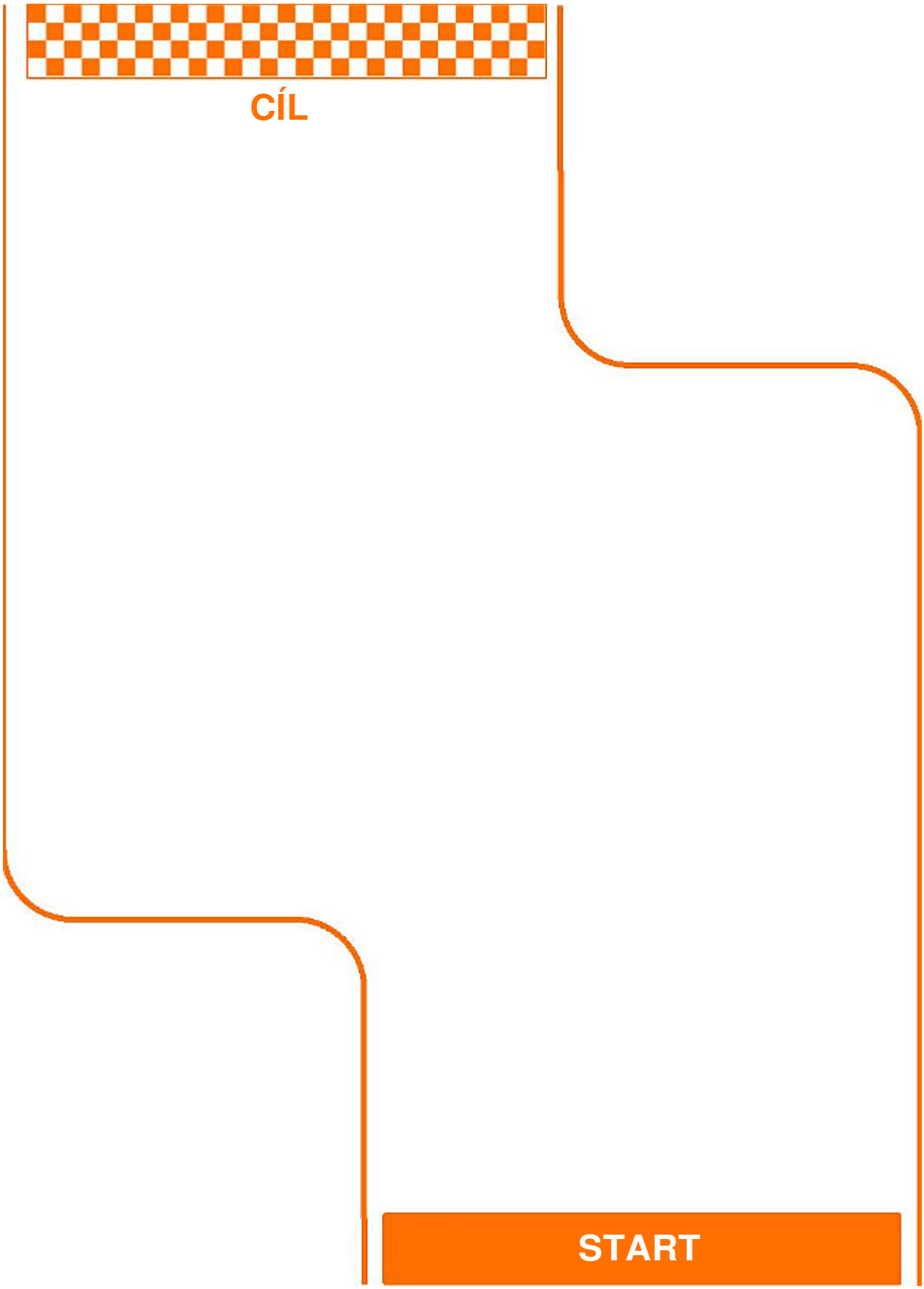
Lekce 3: List aktivity 3.1 – Otočení

Dejte robota Edison na obrys podle obrázku. Vždy spusťte robota od značky Začátek (0°).



Jméno _____

Lekce 3: List aktivity 3.2 – Minibludiště



Jméno _____

Lekce 4: Pracovní list 4.1 – Jízda do čtverce

V této aktivitě je třeba napsat program, který umožní Edisonovi jezdit do čtverce.

Jste na řadě:

Napište program tak, že když Edison jede, dělá přitom čtverec. Použijte příkazy, které jste se již naučili, včetně **Ed**.Drive() a proměnné. Váš program má skončit s Edisonem ve stejném místě, kde jízdu začal.

Stáhněte si svůj program a otestujte jej pomocí listu aktivity 4.1, umístěte Edisona do startovního bodu a jeďte podle čar. Můžete také vytvořit větší čtverec pomocí barevné pásky pro vyznačení čáry a startovního bodu na stole nebo na podlaze.

1. Jak vypadá váš program? Napište kód sem.

_____

_____

_____

_____

_____

_____

_____

2. Kolik volání funkce jste použili v tomto programu?

_____

3. Máte v programu nějaké duplicitní řádky? Pokud ano, které to jsou a kolikrát jste každý z nich použili?

_____

_____

_____

_____

Jméno _____

Lekce 4: Pracovní list 4.2 – Použijte k jízdě do čtverce cyklus

V této činnosti musíte napsat jiný program, který umožní robotovi Edison jezdit do čtverce.

Na pracovním listu 4.1 jste napsali program, který používal stejné příkazy několikrát. Potřebovali jste použít **Ed**.Drive() s parametrem směru **Ed**.FORWARD čtyřikrát, jednou pro každou stranu čtverce. Také jste potřebovali použít **Ed**.Drive() s parametrem směru **Ed**.SPIN_LEFT čtyřikrát, abyste se na každém rohu otočili.

Zjistili jste, že psaní stejných příkazů počád dokola je nudné?

Opakování stejných příkazů znovu a znovu pro počítač není vůbec problémem, ale takový zápis programu není příliš efektivní. Místo toho je lepší použít strukturu cyklu.

Sledujte Marka Zuckerberga, který vytvořil Facebook, jak vysvětluje koncept cyklů při programování:
https://www.youtube.com/watch?v=hYvcoRkAkOU

Kdo věděl, že skvělý kodér se může stát jedním z nejmladších miliardářů světa?
Hodnota čistého jmění Marka Zuckerberga se odhaduje na více než 70,5 miliardy US dolarů!

Můžeme napsat program pro jízdu do čtverce s úspornějším kódem pomocí cyklu **for**. Tím se zvýší efektivita psaní programu. Protože bude stačit méně řádků kódu, použití cyklu **for** také pomůže snížit pravděpodobnost překlepů a chyb syntaxe v programu.

Cyklus 'for' a funkce 'range()' v Pythonu

V Pythonu je cyklus **for** řídící strukturou, kterou lze použít k opakování sady příkazů libovoněkrát.

Pomocí cyklus **for** můžete opakovat (někdy se říká 'iterovat') blok příkazů tolikrát, kolikrát chcete.

Cyklus **for** se v Pythonu často používá společně s funkcí **range()**.

Funkce range() vrátí řadu hodnot v rámci určitého rozsahu.

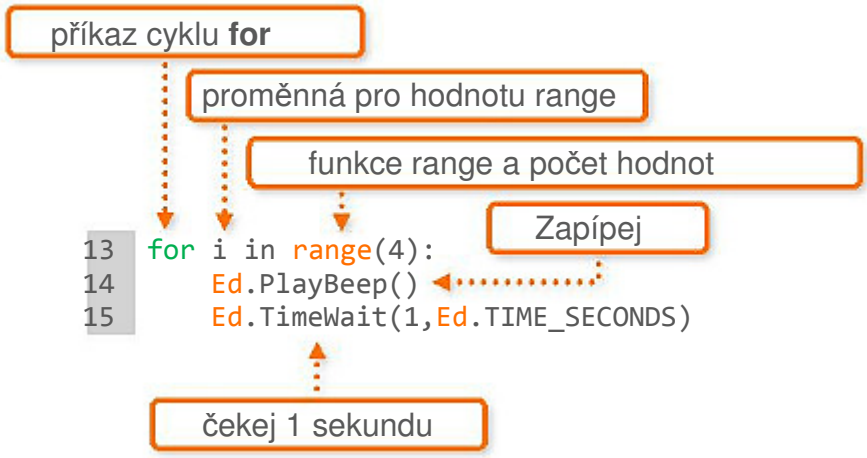
V EdPy má range() jen jeden vstupní parametr. Tento vstupní parametr určuje horní hranici řady a spodní hranice je vždy 0.
[Pozn. př.: Jen jeden vstupní parametr range() není nějaká nedovolená redukce běžného Pythonu, ale použitá jednodušší frma funkce range() je součástí Pythonu a je rovnocenná se složitější verzí range() s více parametry]

Funkce range() vrací hodnoty od **0** do čísla **(vstupní parametr – 1)**.

Příklad:
range(4) → dává 4 hodnoty v řadě: 0, 1, 2, 3.

Jméno _____

Podívejme se na příklad:



V tomto příkladu se cyklus **for** opakuje čtyřikrát, což způsobí, že proměnná 'i' má hodnoty 0, 1, 2 a 3. Pokaždé, když se cyklus opakuje, provede se blok příkazů skládající se z **Ed**.PlayBeep() a **Ed**.TimeWait().

Výsledek? Pípnutí se přehrává čtyřikrát s prodlevou jedné sekundy mezi každým pípnutím.

Jste na řadě:

Napište program pomocí cyklu **for** a funkce **range()** tak, že když robot Edison jede, vytvoří čtvereček. Program by mělo být možné sestavit pouze pomocí dvou funkcí **Ed**.Drive(), jedné pro jízdu vpřed a jedné pro otáčení.

Nezapomeňte zahrnout do cyklu dvojtečku a správné odsazení.

Stáhněte si svůj program a otestujte jej pomocí listu aktivity 4.1, umístěte Edisona do startovního bodu a jeďte podle čar. Zkontrolujte, že váš program skončí s Edisonem ve stejném místě, kde začal. Můžete také vytvořit větší čtverec pomocí barevné pásky pro označení čáry a startovního bodu na stole nebo na podlaze.

1. Jak vypadá váš program? Napište jej sem.

_____

_____

_____

_____

_____

_____

Jméno _____

Lekce 4: Pracovní list 4.3 – Jízda v trojúhelníku a šestiúhelníku

V této činnosti je třeba napsat dva různé programy, aby se robot Edison mohl jezdit v trojúhelníku a pak v šestiúhelníku.

Jste na řadě:

Úkol 1: Jeďte v trojúhelníku

Napište program tak, že když robot Edison pojede, vytvoří trojúhelník.

Stáhněte svůj program a vyzkoušejte jej pomocí listu aktivity 4.2, umístěte Edisona do startovního bodu a jeďte podle čar. Můžete také vytvořit větší trojúhelník – pomocí barevné pásky vyznačte čáry a startovní bod na stole nebo podlaze.

1. Kolikrát jste provedli cyklus **for** pro trojúhelníkový tvar?

_____

Úkol 2: Jeďte v šestiúhelníku

Napište program tak, že když robot Edison pojede, vytvoří šestiúhelník.

Stáhněte svůj program a otestujte jej pomocí listu aktivity 4.3, umístěte Edisona do startovního bodu a jeďte podle čar. Můžete také vytvořit větší šestiúhelník – pomocí barevné pásky vyznačte čáry a startovní bod na stole nebo podlaze.

2. Kolikrát jste provedli cyklus **for** pro šestiúhelníkový tvar?

_____

3. Měli byste odhalit schématický vztah vznikající mezi počtem stran tvaru a počtem opakování cyklu **for**. Popište tento vztah.

_____

_____

4. Kolikrát byste potřebovali provést cyklus **for**, aby se nakreslil pravidelný dvanáctiúhelník (to znamená, že má všechny strany stejné)?

_____

Jméno _____

Lekce 4: Pracovní list 4.4 – Výzva! Jezděte do kruhu

V této aktivitě je třeba napsat program, s kterým bude robot Edison jezdit v kruhu.

Jste na řadě:

Napište program, aby Edison jezdil v kruhu. Edison musí opravdu jezdit dokola, nikoli se otáčet na jednom místě.

Stáhněte si svůj program a vyzkoušejte jej pomocí listu aktivit 4.4, umístěte Edisona do startovního bodu a jeďte podle čáry. Můžete také nechat robota jezdit kolem jakéhokoli kruhovitého objektu, třeba kolem kulatého odpadkového koše nebo stolu.

Rada: Tvar s mnoha sty velmi malých stran se může blížit kružnici.

1. Kolikrát se spustí cyklus, aby výsledný tvar byl kruh?

_____

_____

_____

2. Kolik robot ujede cestuje při každém spuštění cyklu?

_____

_____

3. Jede robot v dokonalém kruhu? Pokud ne, můžete přijít na důvod, proč ne?

_____

_____

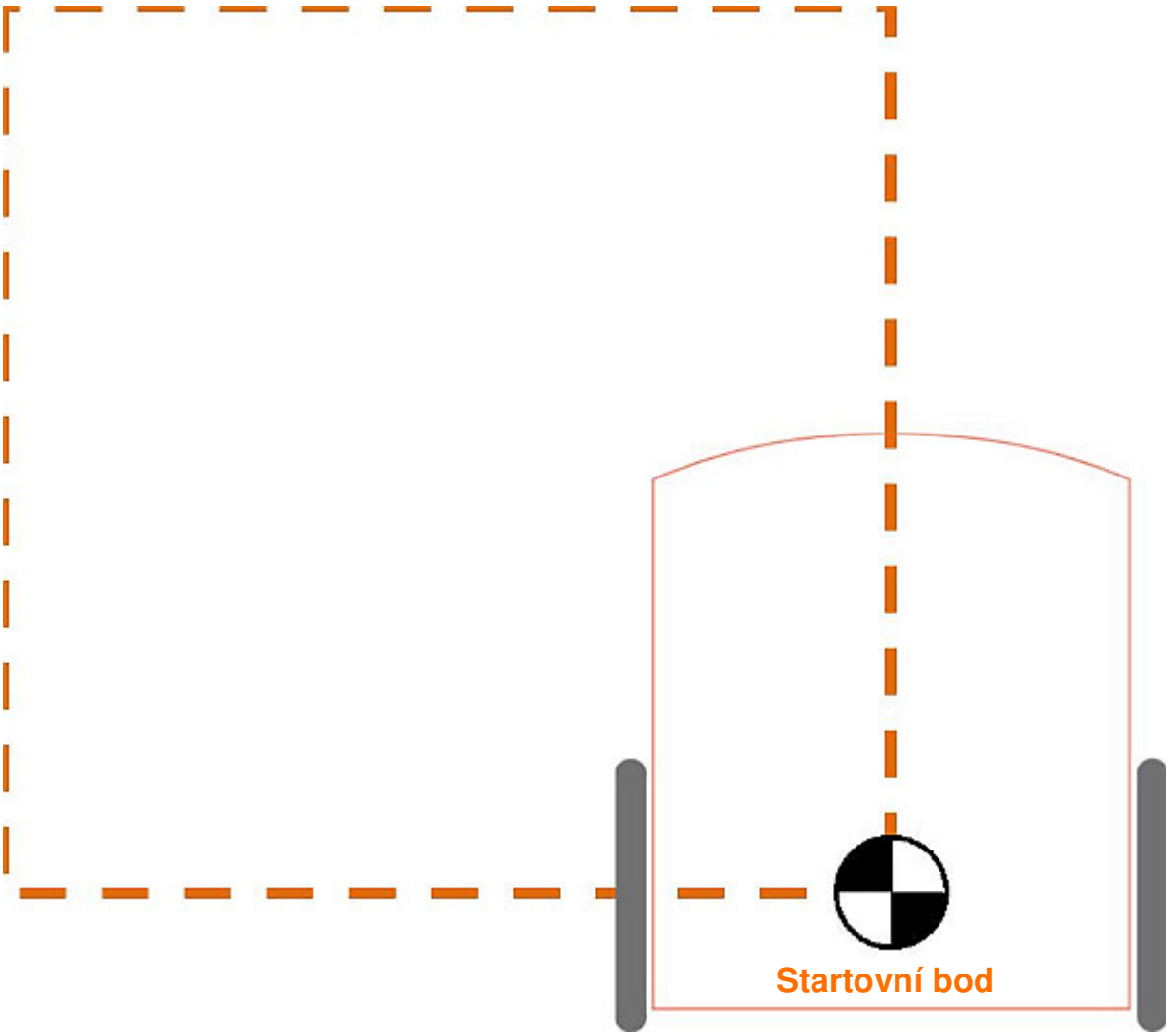
_____

Nepovinná výzva: Nakreslete svůj tvar!

Připojte k robotovi pastelku nebo barevný fix pomocí kombinace bloků LEGO nebo pásky – třeba leukoplasti. Umístěte Edisona na kus papíru a spusťte program pro kruh. Sledujte, jak barevný fix kreslí tvar, když se robot pohybuje. Zjistěte, zda váš Edison kreslí skutečný kruh nebo ne.

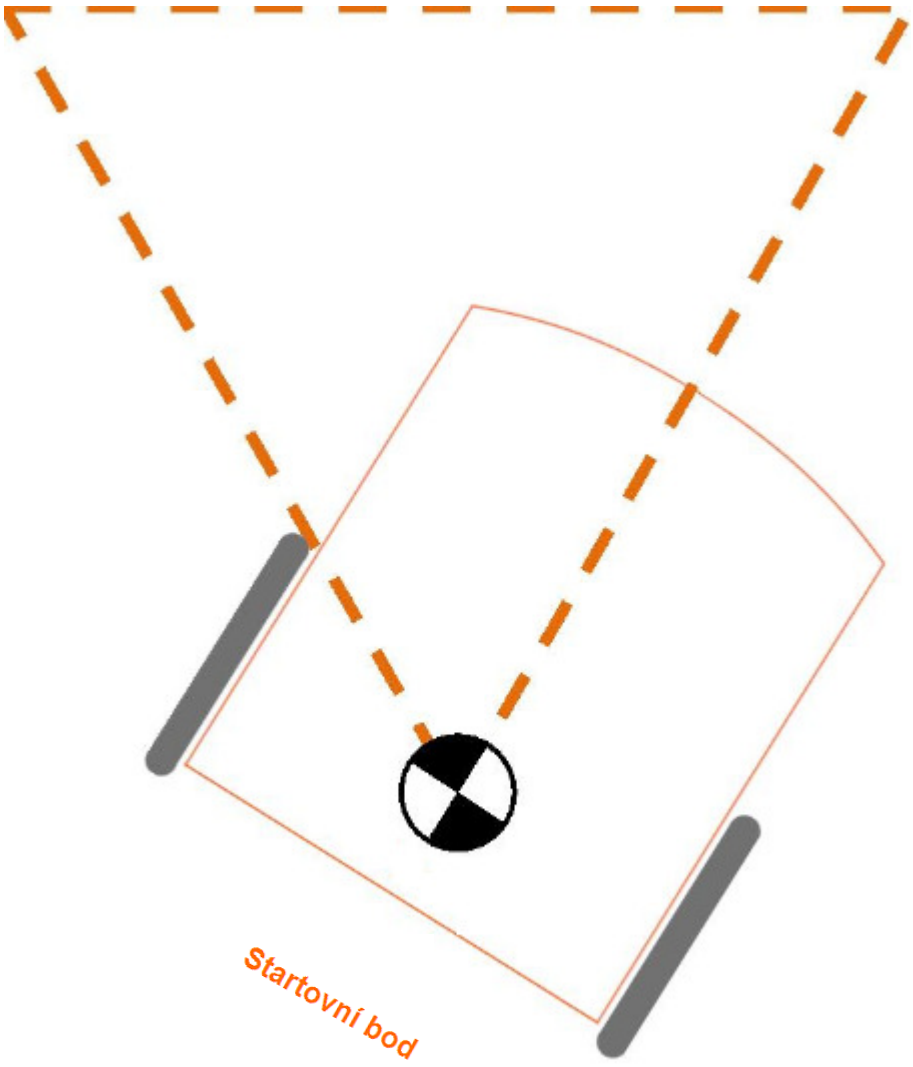
Jméno _____

Lekce 4: List aktivity 4.1



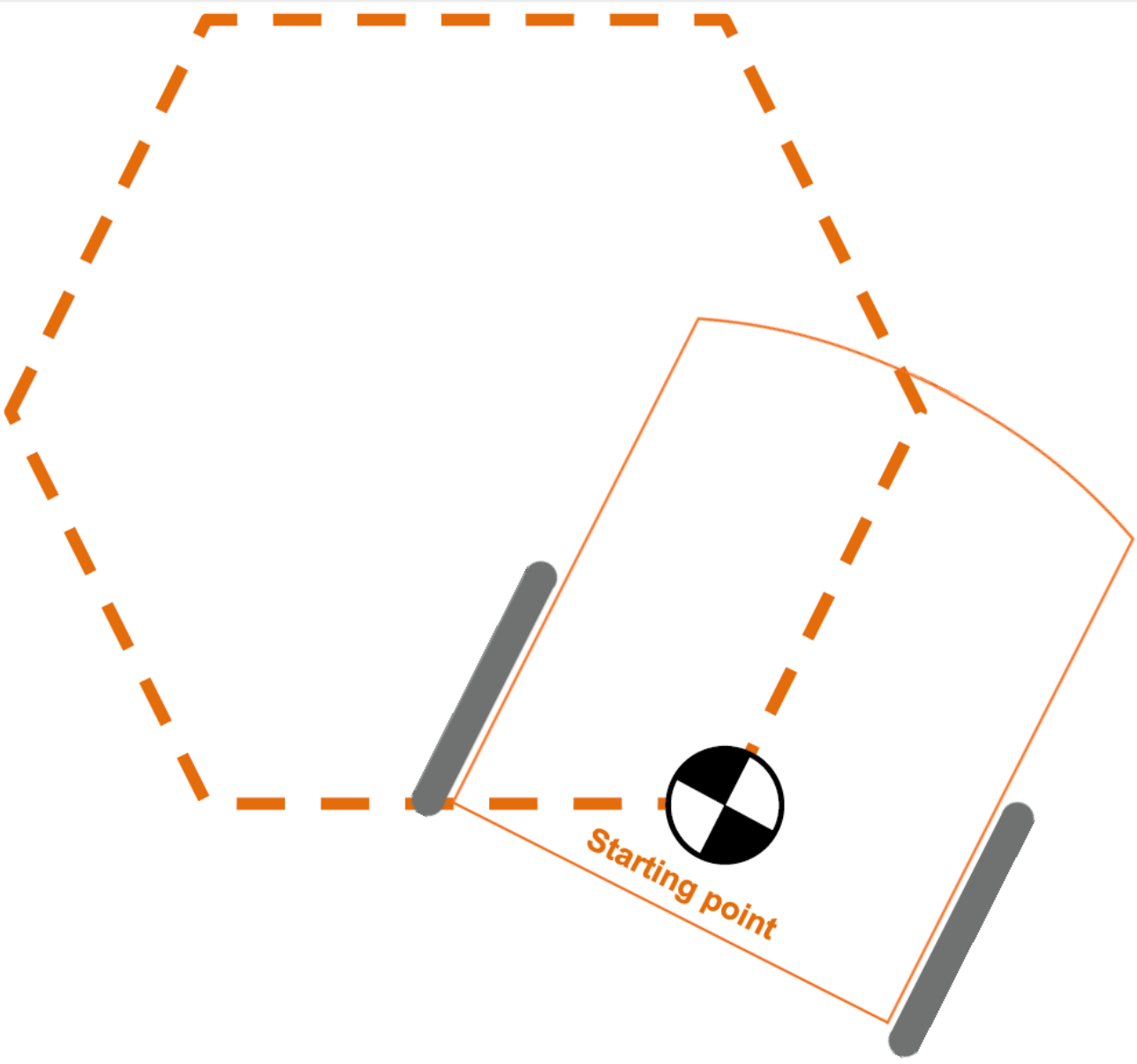
Jméno _____

Lekce 4: List aktivity 4.2



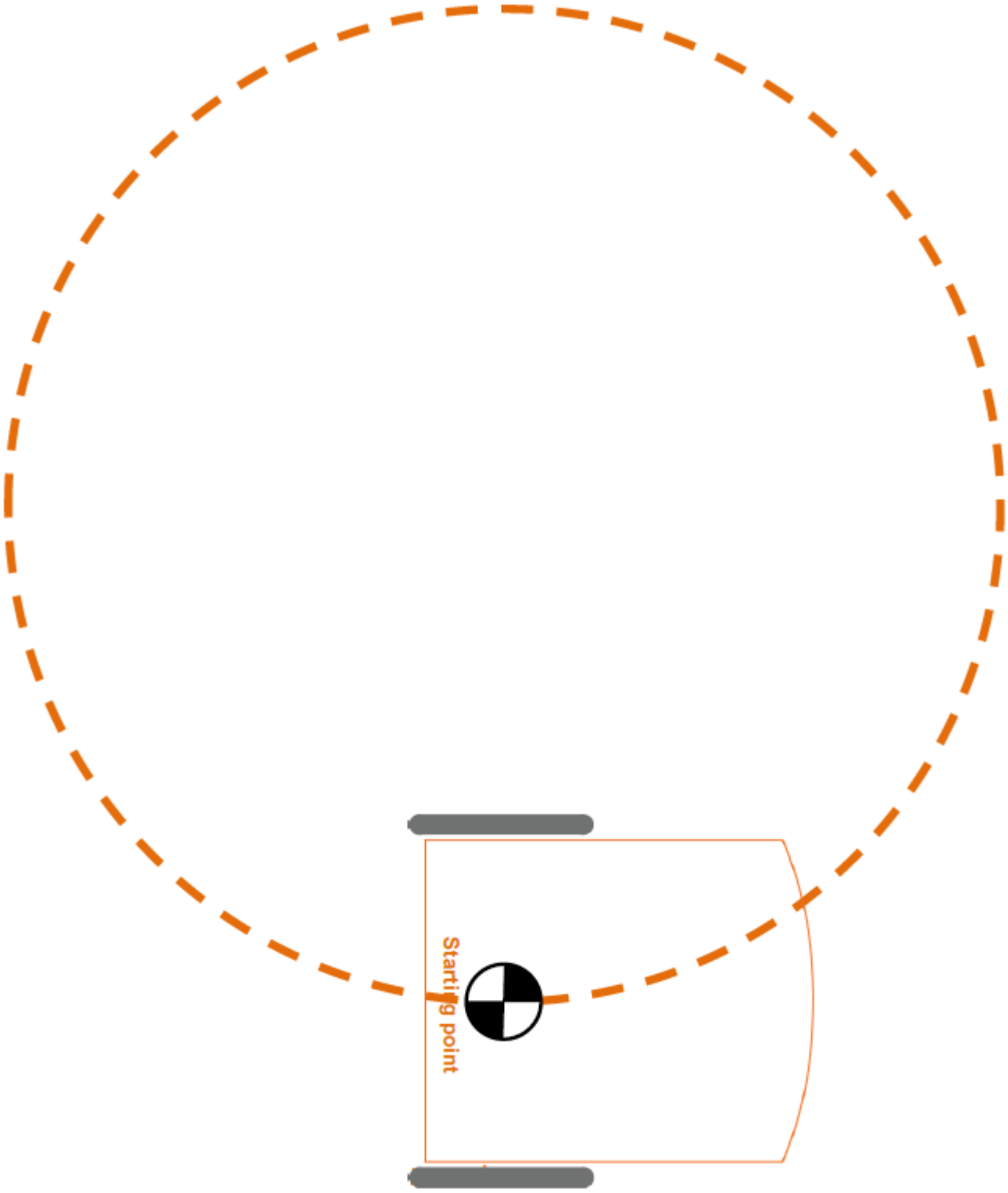
Jméno _____

Lekce 4: List aktivity 4.3



Jméno _____

Lekce 4: List aktivity 4.4



Jméno _____

Lekce 5: Pracovní list 5.1 – Hraní tónů

V této aktivitě máte napsat program, aby Edison zahrál hudební notu/tón a naučíte se, jak Edison v programu přehrává zvuky.

Jednotlivé hudební noty můžete přehrávat pomocí Edisonova malého reproduktoru pomocí funkce **Ed**.PlayTone() v EdPy.

Funkce **Ed**.PlayTone() má dva vstupní parametry: notu a trvání. Nota určuje, jaký tón se má hrát, a délka trvání určuje danou dobu, po kterou by měla být nota zahrána.

Tento seznam obsahuje možné hodnoty vstupních parametrů:

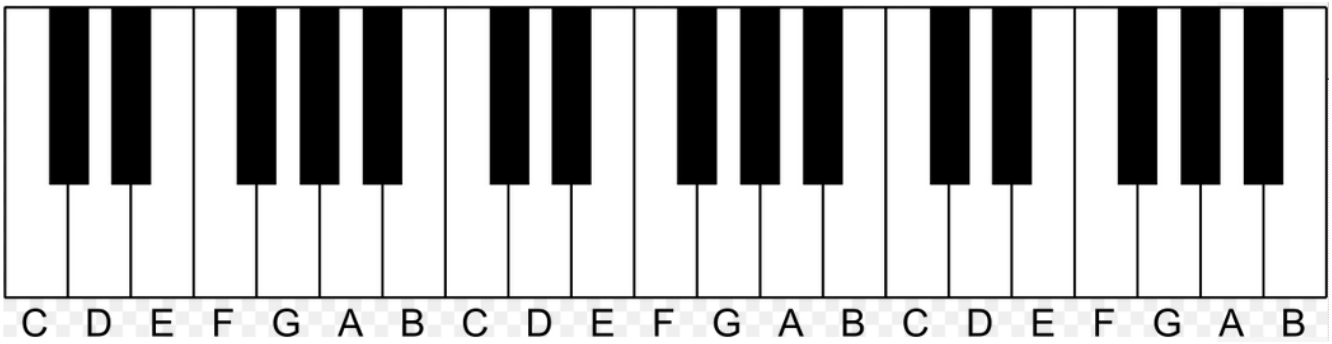
tón/nota

Vstupní parametr	Zahraj notu
Ed.NOTE_A_6	A
Ed.NOTE_A_SHARP_6	Ais
Ed.NOTE_B_6	H
Ed.NOTE_C_7	c
Ed.NOTE_C_SHARP_7	cis
Ed.NOTE_D_7	d
Ed.NOTE_D_SHARP_7	dis
Ed.NOTE_E_7	e
Ed.NOTE_F_7	f
Ed.NOTE_F_SHARP_7	fis
Ed.NOTE_G_7	g
Ed.NOTE_G_SHARP_7	gis
Ed.NOTE_A_7	a
Ed.NOTE_A_SHARP_7	ais
Ed.NOTE_B_7	h
Ed.NOTE_C_8	c'
Ed.NOTE_REST	pomlka

trvání

Vstupní parametr	Přehrávej po
Ed.NOTE_SIXTEENTH	125 millisekund
Ed.NOTE_EIGHTH	250 millisekund
Ed.NOTE_QUARTER	500 millisekund
Ed.NOTE_HALF	1000 millisekund
Ed.NOTE_WHOLE	2000 millisekund

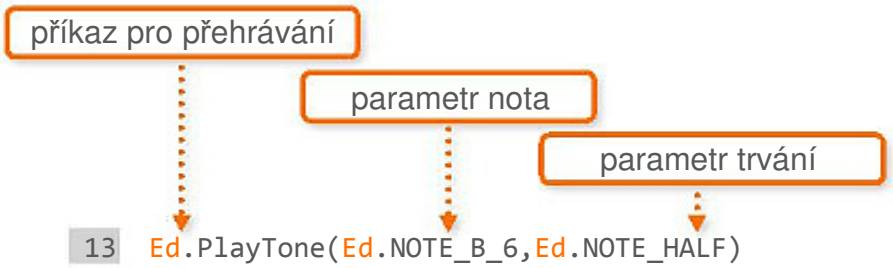
Pozn. př.: České názvy not nesouhlasí přesně s anglickými názvy – viz B/b a H/h.



*Obrázek **anglické klaviatury**. [Zdroj anglická Wikipedie.]*

Jméno _____

Podívejme se blíže na funkci přehrávání v programu:



Můžete zjistit, co bude program dělat? (Použijte údaje z tabulky hodnot parametrů.)

Tento program bude hrát tón H po dobu 1 sekundy.
(Nezapomeňte, že anglické B je české H)

Jste na řadě:

Úkol 1: Přehrajte notu

Napište následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 Ed.PlayTone(Ed.NOTE_A_SHARP_7,Ed.NOTE_HALF)
14
```

Stáhněte a otestujte program, abyste zjistili, jak to zní.

Úkol 2: Přehrajte tón a pak jeďte. Nebo hrajte během jízdy?

Když Edison přehrává zvuky, dělá to "na pozadí". To znamená, že jakmile Edison začne přehrávat zvuk, program se přesune na další řádek kódu. Zvuk bude pokračovat v přehrávání na pozadí a během toho Edison pokračuje v programu.

Pokud chcete, aby Edison čekal, až zvuk skončí, musíte použít funkci **Ed**.ReadMusicEnd() v cyklu **while**.

Jméno _____

Napište následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 Ed.PlayTone(Ed.NOTE_C_8, Ed.NOTE_WHOLE)
13 while Ed.ReadMusicEnd()==Ed.MUSIC_NOT_FINISHED:
14     pass
15 Ed.Drive(Ed.FORWARD, Ed.SPEED_6, 5)
16
```

Stáhněte a otestujte program.

1. Popište, co se stalo při spuštění tohoto programu.

_____

_____

_____

2. Podívejte se na řádky 13 a 14 programu. Pamatujte, že výrazy porovnávají levou stranu s pravou stranou zápisu ve výrazu. Co dělá tento cyklus?

_____

_____

_____

Jméno _____

Napište následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 Ed.PlayTone(Ed.NOTE_C_8, Ed.NOTE_WHOLE)
13 Ed.Drive(Ed.FORWARD, Ed.SPEED_6, 5)
14
```

Stáhněte a otestujte program.

3. Popište, co se stalo při spuštění tohoto programu.

_____

_____

_____

4. Proč se tento program choval jinak než předchozí program?

_____

_____

_____

Jméno _____

Lekce 5: Pracovní list 5.2 – Vytvoření alarmu

V této aktivitě máte napsat program, aby Edison spustil poplach na zvolené frekvenci.

Pomocí funkce **Ed**.PlayTone() můžete pomocí čísel a proměnných přizpůsobit přesnou frekvenci zvuku, který Edisonův reproduktor produkuje.

Frekvence v akustice

Jak možná víte, zvuk se šíří ve vlnách zvaných zvukové vlny. Akustika, obor fyziky, která se zabývá zvukem a zvukovými vlnami, se zabývá všemi záležitostmi kolem zvuku, včetně toho, jak zvuk měřit.

Jedním ze způsobů měření zvuku je měření frekvence. Frekvence je počet vln procházejících jedním bodem za určitý časový úsek.

Frekvence se nejčastěji měří v cyklech za sekundu (cyklus/s). Základní jednotka pro frekvenci je hertz, zkracuje Hz (čti 'herc').

Jeden hertz se rovná jedné úplné vlně za sekundu - je to právě jeden cyklus za sekundu.

Věděli jste? Rozsah lidského sluchu je asi 20 Hz ~ 20000 Hz.

Frekvence a doba

Kromě hudebních not, které jsou v EdPy přednastaveny, můžeme také Edisona naprogramovat k přehrávání zvuků s různými frekvencemi.

Za tímto účelem převádíme frekvenci na periodu, které Edison rozumí.

Perioda je doba dokončení jednoho plného cyklu. Protože používáme hertz, měříme frekvenci v cyklech za sekundu. Perioda je pak 1/frekvence. Když se perioda zvětšuje, frekvence klesá.

Podívejme se na některé příklady toho, v jakém vztahu je frekvence a perioda:

- Pokud vlna trvá 0,5 sekundy, má frekvenci 2 Hz, protože může dokončit 2 cykly za 1 sekundu.
- Pokud má vlna periodu 2 sekundy, má frekvenci 0,5 Hz, protože za 1 sekundu může dokončit pouze polovinu cyklu.

Převod frekvence na periodu pro program v EdPy

Chcete-li, aby Edison hrál vlastní frekvenci, musíme vyčíslit hodnotu periody. Toto je číslo, které zadáváme do parametru 'nota' v **Ed**.PlayTone().

Chcete-li převést frekvenci na perodu, dělte číslo 8 000 000 požadovanou frekvencí. Např. pro přehrávání zvuku s frekvencí 1kHz (1000 cyklů za sekundu). (Pozn. př.: Číslo 8000000 je dáno vnitřní konstrukcí Edisona.)

$$\frac{8000000}{1000} = 8000$$

Jméno _____

Jste na řadě:

Úkol 1: Přehrajte vlastní tón

Napište následující program:

Stáhěte a otestujte, abyste slyšeli, jako co zní tento program.

Úkol 2: Přehrajte poplach (alarm)

V tomto programu chceme, aby Edison hrát noty s rostoucí periodou.

Chcete-li udělat poplachový program, musíte použít cyklus **for**, proměnné a funkci **range()**. Také je třeba vložit do programu cyklus **while**.

Napište následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 for i in range(33):
14     Ed.PlayTone(100+(i*100), 1000)
15     while Ed.ReadMusicEnd()==Ed.MUSIC_NOT_FINISHED:
16         pass
17
```

Stáhněte jej a vyzkoušejte, jako co tento program zní.

1. Co slyšíte od robota? Proč se tohle děje?

_____

_____

_____

Jméno _____

Důležitá dovednost v programování je schopnost tzv. trasovat (sledovat) program, aby programátor pochopil, co se děje. Programátoři provádějí trasování kódu jako metodu pro ruční simulaci provedení kódu, aby ověřili, zda funguje správně, než jej přeloží.

Trasování znamená procházení programu řádek za řádkem, zaznamenávání důležitých hodnot. Často se to dělá, když se hledají chyby, někdy se říká "mouchy" v kódu, ale trasování je také užitečné, když potřebujete pochopit, co se v programu děje.

Pokuste se 'trasovat' to, co se děje v programu, a odpovězte na následující otázky o programu.

2. Vyplňte následující tabulku výpočtem parametru perioda pro každou danou hodnotu "i" ve výše uvedeném kódu. První hodnota je už vyplněna.

Hodnota **i**	Parametr perioda [první vstupní parametr do PlayTone()]
0	100
1	
2	

3. Jaká je maximální hodnota **i**?

_____

4. Jaká je maximální hodnota vstupního parametru do funkce PlayTone()?

_____

5. Kolik tónů se přehraje?

_____

Zkuste to!

Aplikace akustiky v technice se nazývá akustické inženýrství.

Zkuste nějaké vlastní akustické inženýrství. Experimentujte s úpravou parametrů PlayTone(), aby program přehrával jinou kombinaci zvuků.

Jméno _____

Lekce 5: Pracovní list 5.3 – Přehrávání melodie

V této aktivitě máte napsat program, aby Edison hrál hudební melodii.

Edisona můžete přimět k hraní pomocí funkce Ed.PlayTune() a nového vstupního typu dat nazvaného **string** (=řetězec).

Pozn.př.: Časté české překlady slova string jsou řetězec nebo výjimečně řetěz. Zde ale důsledně jako už dříve zůstaneme u anglického pojmu všude tam, kde je to nutné v různých příkladech. Překladem klíčových slov programu se často nadělá víc škody než užitku. Výjimku tvoří ***výklad pojmů*** např. v této i jiných učebnicích. *Pro úplnou přesnost a do budoucna:* Klíčové slovo Pythonu pro string je `str` a samotný **string** je objekt.

Použijte string pro přehrání melodie

V Pythonu je **string** uspořádaný seznam znaků. **Character** (=znak) je vše, co můžete napsat na klávesnici jako písmeno, číslo, nebo zvláštní znak jako je $ nebo #. Např. 'Meet Edison' je řetězec dlouhý 11 znaků (10 písmen a 1 mezera).

Pozn. př. *Pro úplnou přesnost a do budoucna:* Klíčové slovo Pythonu pro **character** neexistuje, datový typ character neexistuje samostatně. Za character (znak) považujeme string délky jedna.

V aplikaci EdPy potřebujeme použít k přehrávání hudební melodie zvláštní řetězec. Říkáme mu našim Edisonovým názvem 'řetězec melodie'. Řetězec melodie je speciální řetězec znaků, který představuje jednu konkrétní melodii. Řetězce melodie se skládají z not a jejich trvání, které jsou vyjádřeny jednotlivými znaky.

Řetězec melodie vypadá takto: "ndndndndnd...ndz", kde **n** je hudební nota z tabulky not a **d** je její trvání z tabulky trvání, **z** je koncový znak:

Tabulka not

Znak řetězce	Zahraje notu
m	A
M	Ais
n	H
c	c
C	cis
d	d
D	dis
e	e
f	f
F	fis
g	g
G	gis
a	a
A	ais
b	h
o	c'
R	pomlka
z	konec melodie

Tabulka trvání

Znak řetězce	Přehraje se
1	nota celá
2	nota půlová
4	nota čtvrťová
8	nota osminová
16	nota šestnáctinová

Jméno _____

Všechny řetězce melodie musí korektně končit znakem 'z'.

Chcete-li vytvořit řetězec melodie, musíte zavolat funkci **Ed**.TuneString(), která má dva vstupní parametry. Velikost řetězce (jinými slovy počet znaků v řetězci) je první parametr a aktuální řetězec, který chcete přehrát, je druhý parametr.

Rychlost přehrávání melodie můžete změnit změnou proměnné **Ed**.Tempo v nastavovací části programu.

Jste na řadě:

Napište následující kód pro přehrání melodie "Mary měla jehňátko"

```
1
2  #-------------Setup---------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 maryLamb = Ed.TuneString(53,"e4d4c4d4e4e4e2d4d4d2e4g4g4e4d4c4d4e4e4e4e4d4d4e4d4c1z")
13
14 Ed.PlayTune(maryLamb)
15 while Ed.ReadMusicEnd() == Ed.MUSIC_NOT_FINISHED:
16     pass
17
```

Řetězec melodie v tomto programu je:

"e4d4c4d4e4e4e2d4d4d2e4g4g4e4d4c4d4e4e4e4e4d4d4e4d4c1z"

Experimentujte se změnou hodnoty **Ed**.Tempo v nastavovací části programu.

1. Jaké jsou různé hodnoty, které **Ed**.Tempo může mít?

Rada: Nezapomeňte, že v EdPy můžete použít funkci automatického našeptávání. Zkuste zadat **Ed**.TEMPO a uvidíte všechny možné hodnoty pro **Ed**.TEMPO, které našeptávač ukáže.

_____

_____

2. Která hodnota **Ed**.TEMPO melodii přehraje nejrychleji?

_____

Jméno _____

3. Změňte svůj program tak, aby přehrával pouze část melodie. Popište změny, které jste museli v programu provést, aby hrál jen část melodie.

_____

_____

_____

_____

_____

Jméno _____

Lekce 5: Pracovní list 5.4 – Nechejte robota tančit

V této aktivitě napíšete program, aby váš robot tančil.

Ve většině dobrých tanečních vystoupeních jsou některé kroky či akce, které se opakují. Akce Edisona v taneční sestavě můžete opakovat pomocí cyklu **for**.

"Shimmy" (čti šimi) je taneční pohyb, kdy tělo stojí a ramena se rychle pohybují dopředu a dozadu (když jde levé rameno vpřed, jde pravé vzad a opačně).

Podívejte se na následující program, kdy Edison předvede verzi shimmy:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 #Set up some variables
14 turnSpeed = Ed.SPEED_9
15 degreesToTurn = 20
16 numberOfTwists = 3
17
18 #Now shimmy!
19 Ed.Drive(Ed.SPIN_RIGHT,turnSpeed,degreesToTurn/2)
20 for i in range(numberOfTwists):
21     Ed.Drive(Ed.SPIN_LEFT,turnSpeed,degreesToTurn)
22     Ed.Drive(Ed.SPIN_RIGHT,turnSpeed,degreesToTurn)
23 Ed.Drive(Ed.SPIN_LEFT,turnSpeed,degreesToTurn/2)
24
```

Tento program používá proměnné tak, aby bylo snadné změnit rychlost otáčení, počet obrátek v tanci a úhlové stupně *[degress]*, o které se Edison pootočí.

Řádky 13 a 18 začínají znakem **#**, to znamená, že tyto řádky jsou řádky poznámek v programu, které nám usnadňují čtení programu. Zapamatujte si, že Edison vynechá všechny řádky, které začínají znakem **#**.

Podívejte se na řádky 19 a 23. Na těchto řádcích dělá program matematický výpočet, aby se Edison obrátil jen o polovinu zadaných úshlových stupňů.

Jste na řadě:

Napište program.

Stáhněte si program do Edisona a spusťte jej, abyste viděli tanec v akci.

Jméno _____

1. Kolikrát se robot otočí doleva?

_____

2. Kolikrát se robot otočí doprava?

_____

3. První otočení doprava je pouze polovina vzdálenosti všech otáček uvnitř cyklu **for**, protože tento řádek má vstupní parametr 'degreesToTurn/2'. Proč máte tento řádek v programu? Zkuste odstranit matematiku (/2) a spusťte program znovu. Čeho si všimnete? (*Rada:* podívejte se, jak daleko se Edison pohne doleva ve srovnání se startovním bodem.)

_____

_____

_____

_____

Zkuste to!

Experimentujte s programem. Zkuste změnit proměnné, abyste změnili způsob, jakým Edison tančí. Změňte počet stupňů, o které se Edison otočí, rychlost s jakou se Edison otáčí, počet obratů v tanci nebo změňte všechny tři parametry!

Jméno _____

Lekce 5: Pracovní list 5.5 – Výzva! Tanec na hudbu

Tanec je zábavnější s hudbou! V této aktivitě napíšete program kombinující taneční pohyby s některými tóny nebo melodií.

Jste na řadě:

Napište a spusťte následující program, který kombinuje tanec shimmy s některými tóny:

```
#------------Setup---------------

import Ed

Ed.EdisonVersion = Ed.V2

Ed.DistanceUnits = Ed.CM
Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

#Set up my variables
turnSpeed = Ed.SPEED_9
degreesToTurn = 60
numberOfTwists = 3

#Now dance to the music!
Ed.Drive(Ed.SPIN_RIGHT, turnSpeed, degreesToTurn/2)
Ed.PlayTone(Ed.NOTE_C_7, Ed.NOTE_SIXTEENTH)
for i in range(numberOfTwists):
    Ed.Drive(Ed.SPIN_LEFT, turnSpeed, degreesToTurn)
    Ed.PlayTone(Ed.NOTE_A_7, Ed.NOTE_SIXTEENTH)
    Ed.Drive(Ed.SPIN_RIGHT, turnSpeed, degreesToTurn)
    Ed.PlayTone(Ed.NOTE_C_7, Ed.NOTE_SIXTEENTH)
Ed.Drive(Ed.SPIN_LEFT, turnSpeed, degreesToTurn/2)
Ed.PlayTone(Ed.NOTE_A_7, Ed.NOTE_SIXTEENTH)

```

Nyní navrhněte pro Edisona svůj vlastní tanec přidáním některých tónů nebo použitím melodie. Můžete vše synchronizovat tak, že Edison tančí souhlasně s hudbou?

1. Popište taneční pohyby svého robota. Je ve vašem programu něco, co se vám opravdu líbí? Pokud ano, popište to.

_____

_____

_____

_____

_____

Jméno _____

2. Jakou kombinaci tónů nebo hudebních not jste hráli společně s tancem?

_____

_____

_____

_____

Jméno _____

Lekce 6: Pracovní list 6.1 – Bliknutí diody LED v reakci na tlesknutí

V této činnosti je třeba napsat program, který využije čidlo Edisona, které umí detekovat tlesknutí, aby robot zablesknul jednou LED diodou, kdykoliv se tleskne.

První věcí je program naplánovat!

Vývojové diagramy v programování

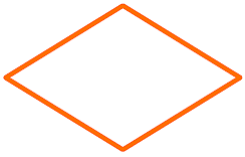
Profesionální programátoři než začnou psát svůj kód, program obvykle plánují/projektují. Použití **vývojových diagramů** je jeden ze způsobů, jak mohou programátoři organizovat a naplánovat své programy.

Pozn. př.: Jiné způsoby záznamu jsou ekvivalentní, zmíním jen dva z nich: strukturogramy a UML = Unified Modeling Language. UML je dnes faktickým standardem a velmi doporučuju budoucím programátorům jeho samostudium.

Myšlenkou vývojového diagramu je graficky shrnout, co se v programu děje, aniž by bylo nutné jít do všech podrobností. Vývojové diagramy umožňují programátorovi vizualizovat a sdělovat, jak bude fungovat "tok" programu.

Ve vývojovém diagramu je program znázorněn různými tvary a šipkami. Každý tvar představuje jiný prvek programu a šipky ukazují, jak tyto prvky spolupracují.

Ve vývojových diagramech je použito pět hlavních symbolů:

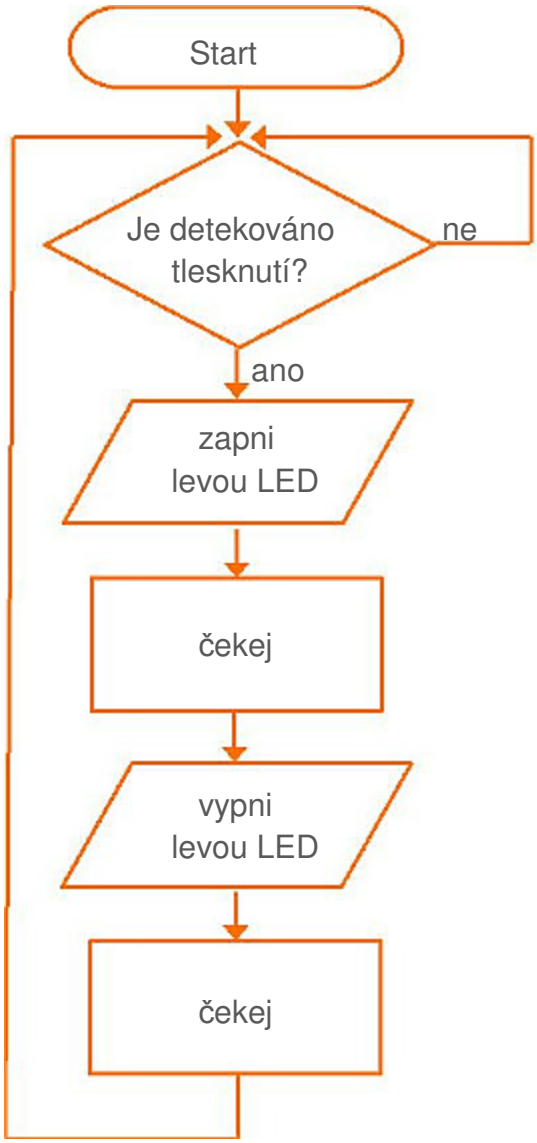
Symbol	Název	Funkce
	Koncový symbol (začátek/konec)	Začátek a konec programu (algoritmu)
	Šipka	Čára znázorňující spojení/vztahy mezi tvary
	Vstup/výstup	Lichoběžník znamená vstup nebo výstup z programu
	Zpracování	Příkaz / proces / akce
	Rozhodování (podmíněný příkaz)	Kosočtverec znázorňuje rozhodnutí

Jméno _____

Složitější vývojové diagramy mohou používat i další tvary s různými významy.

Při vytváření vývojového diagramu se často dovnitř tvarů nebo vedle šipek připisují slova. Tato slova jsou krátká stručná vyjádření ke zpracování nebo rozhodování.

Podívejme se na příklad vývojového diagramu, který stručně vyjadřuje program, který chceme udělat. Zde je vývojový diagram programu, který robotovi řekne, že má čekat na tlesknutí a pak blikne levou LED diodou:



Tento program použije čidlo Edisona pro detekci zvuku, aby zjistil, zda došlo k tlesknutí. Výsledkem zjišťování (rozhodování) je další tok programu (kudy porogram pokračuje).

Když se na tento vývojový diagram podíváte, možná si všimnete, že nemá koncový symbol 'konec'. Je to proto, že program používá cyklus **while** nastavený tak, aby program stále pokračoval.

Jméno _____

Vytvoření nekonečné smyčky – *shrnutí*

Někdy můžete chtít napsat program, který nemá konec, ale pracuje navždy stále dokola. V programování se toto často označuje jako nekonečná smyčka/cyklus.

Také ve vývojovém diagramu můžete vytvořit nekonečnou smyčku.

Prohlédněte si následující program, který odpovídá našemu předchozímu vývojovému diagramu:

```
1
2  #------------Setup---------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 while True:
13     Ed.ReadClapSensor()
14     while Ed.ReadClapSensor() != Ed.CLAP_DETECTED:
15         pass
16     Ed.LeftLed(Ed.ON)
17     Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
18     Ed.LeftLed(Ed.OFF)
19     Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
20
```

Tento program obsahuje nekonečnou smyčku/cyklus.

Podívejte se na řádek 12 programu, kde se použil cyklus **while**.

Cyklus **while** vždy potřebuje podmínku. Když tato podmínka dává hodnotu **true**, cyklus provede libovolný kód v těle cyklu (zde v Pythonu je to ten odsazený kód).

Chceme-li cyklus **while** opakovat nekonečněkrát, místo podmínky, kterou by musel program pokaždé vyhodnocovat nějakým porovnáváním, napíšeme jednoduše slovo True. (True je dopravdy pevně nastavený, konstantní objekt v Pythonu, pozn. př.)

True se vždy vyhodnotí jako **true**. ☺ ☺ Nastavením podmínky na hodnotu True jsme ***pevně zakódovali (natvrdo naprogramovali)*** podmínku našeho cyklu **while** na hodnotu **true**.

V programování jako ***pevné zakódování*** nazýváme skutečnost, že si něco specifického a hlavně stálého vynutíme tím, že to do programu výslovně a jasně napíšeme.

A jinými slovy si ještě jednou zopakujeme totéž: Použitím True jako podmínky pro cyklus **while** v tomto programu navždy platí, že podmínka cyklu se už nikdy nemůže vyhodnotit jako **false**, a cyklus se proto bude opakovat donekonečna.

Jméno _____

Jste na řadě:

Napište výše uvedený program, který ovládá robota Edisona tak, aby levý indikátor LED bliknul po tleknutí. Program stáhněte a vyzkoušejte, jak funguje.

1. Jaká je nejdelší vzdálenost mezi vámi a Edisonem, aby robot ještě detekoval vaše tlesknutí?

_____

2. Jaký účel má volání funkce TimeWait() v kódu? Co by se stalo, kdybychom je neměli? *Rada:* Zkuste spustit program bez volání funkce TimeWait ().

_____

_____

_____

3. Podívejte se na program a vývojový diagram, abyste porovnali, jaký je vzájemný vztah programu a vývojového diagramu. Vysvětlete, co se v kódu stane, když výsledek rozhodnutí znázorněný ve vývojovém diagramu je "ne".

_____

_____

_____

Zkuste to!

Můžete změnit program tak, aby se na tlesknutí rozsvítily obě LED?

Jméno _____

Lekce 6: Pracovní list 6.2 – Jeďte v reakci na tlesknutí

V této aktivitě máte napsat program, kdy se Edison v reakci na tlesknutí rozjede dopředu.

Jste na řadě:

Úkol 1: Když se detekuje tlesknutí (angl. CLAP), jeďte dopředu

Napište a spusťte následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 Ed.ReadClapSensor()
13 while Ed.ReadClapSensor() == Ed.CLAP_NOT_DETECTED:
14     pass
15 Ed.Drive(Ed.FORWARD,Ed.SPEED_8,10)
16
```

1. Proč je nutné provést na řádku 12 počáteční čtení snímače tlesknutí? Co to udělá?
Rada: Podívejte se zpátky na pracovní list 2.5.

_____

_____

_____

Úkol 2: Jděte dopředu, a když je zjištěno tlesknutí, jeďte dozadu,

Zvukové čidlo robota Edison není citlivé jenom na tlesknutí. Čidla mohou reagovat na jakýkoli hlasitý zvuk, a proto můžete spustit zvukové čidlo klepnutím poblíž reproduktoru robota,

Motory, převody a kola Edisona při otáčení vydávají zvuk, což může spustit zvukové čidlo. Chcete-li zabránit tomu, aby zvuk jízdy robota spouštěl zvukové čidlo, je třeba program změnit.

Budete muset přidat volání funkce TimeWait() se vstupním parametrem přibližně 350 milisekund, aby se motory robota stačily zastavit.

Také je potřeba pro vymazání čidla tlesknutí použít funkci ReadClapSensor().

Jméno _____

Napište následující program:

```
#-------------Setup----------------

import Ed

Ed.EdisonVersion = Ed.V2

Ed.DistanceUnits = Ed.CM
Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------
Ed.Drive(Ed.FORWARD,Ed.SPEED_8,10)

Ed.TimeWait(350,Ed.TIME_MILLISECONDS)
Ed.ReadClapSensor()
while Ed.ReadClapSensor() == Ed.CLAP_NOT_DETECTED:
    pass
Ed.Drive(Ed.BACKWARD,Ed.SPEED_8,10)

```

Program stáhněte a otestujte.

2. Obvykle si nakreslíme vývojový diagram, který nám pomůže program naplánovat. Pro procvičení nakreslete vývojový diagram, který odpovídá právě napsanému kódu. Nakreslete svůj vývojový diagram níže nebo se pokuste použít program, jako je např. Google Slides (Prezentace Google), pro vytvoření vývojového diagramu.

Jméno _____

Lekce 6: Pracovní list 6.3 – Navrhněte svou vlastní funkci

V této aktivitě navrhnete svou vlastní funkci a použijete ji k napsání Edisoncova programu.

Co přesně jsou funkce?

Nyní, když jste už s Edisonem a EdPy nějakou dobu programovali, tak jste použili řadu různých funkcí z knihovny EdPy.

Jak víte, funkce je kousek kódu, který v programu provádí určitou roli nebo úlohu v závislosti na použitých vstupních parametrech.

Možná jste si však neuvědomili, že všechny funkce, které jste dosud používali, skutečně provedly více řádků kódu, když je program spustil (česky se říká, že program funkci zavolal).

To proto, že funkce je blok uspořádaného, opakovaně použitelného kódu, který se používá k provedení jedné související akce.

Funkce jsou velmi užitečné, protože nám umožňují dobře programovat modulárním způsobem pomocí stejného bloku kódu na různých místech programu. Chcete-li programem spustit všechny tyto řádky bloku kódu ve funkci, stačí zadat jeden řádek: *Volání této funkce*.

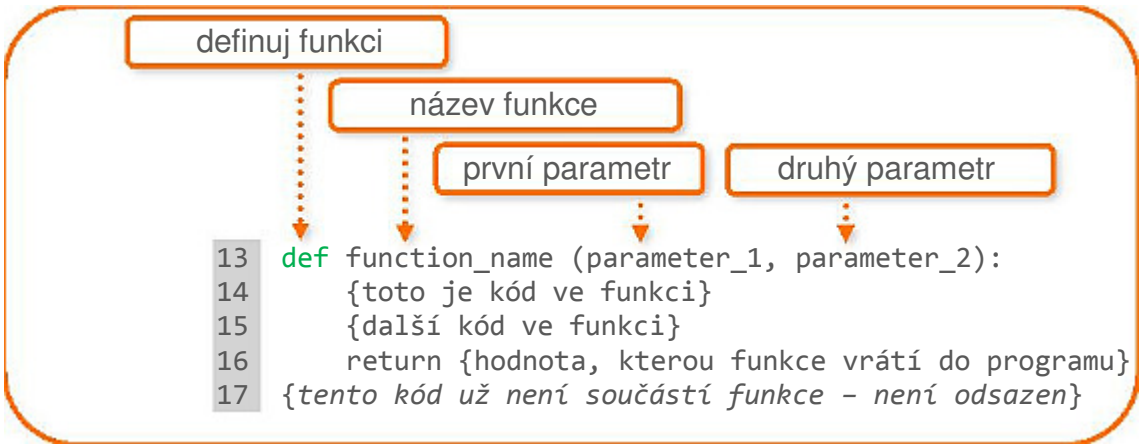
Funkce zjednodušují a zrychlují psaní opětovně využitelného kódu programu.

> Sledujte krátké video podle **code.org**, které vysvětluje, proč jsou funkce v programování tak užitečné: https://youtu.be/8T5acEwfJbw

Vytvoření vlastní funkce

Zatím pokaždé, když jste v EdPy použili funkci, vyvolali jste ji z knihovny **Ed**. Můžete ale také vytvořit vlastní funkce.

V Pythonu funkce vypadají takto:



Vše odsazené patří do definice funkce. Vše, co není odsazeno, není součástí funkce, ale je to běžný další řádek kódu programu.

Je důležité si uvědomit, že funkce, které vytvoříte, se ničím neliší od funkcí, které jste až dosud používali z knihovny **Ed**.

Když voláte funkci (s parametry nebo bez parametrů), program na chvíli odskočí z místa volání funkce na vnitřní kód funkce (je to vlastně ten odsazený kód z předchozího obrázku, řádky 14, 15, 16). Program spustí (vykoná/provede) tento kód a pak se vrátí na řádek, ze kterého jste funkci zavolali.

Pokud určíte, že funkce má vrátit hodnotu, tak když se program po vykonání vnitřního kódu funkce vrátí na řádek běžného programu, odkud jste funkci volali, tak volání funkce se jakoby nahradí hodnotou, kterou funkce vrátila *(někdy třeba vypočítala, pro snažší představu)*. Zde v příkladu ale takové volání a návrat hodnoty funkce neukazujeme.

Je důležité něco vědet o pořadí práce programu s jednotlivými elementy: 1. program nejprve vyřeší funkce, 2. pak matematické příkazy, 3. pak výrazy.

Uspořádání programu

Konvence v nástroji EdPy spočívá v zapsání definic vašich vlastních funkcí na konci kódu i poté, co jste své funkce už použili (volali) ve svém programu. Pozn. př.: Zní to neobvykle, ale ničemu to nevadí, překladač si s takovou situací bez problémů poradí a "dobře ví", kde má hledat ty kousky kódu, o které nám ve vlastních funkcích jde.

Tak se dosáhne toho, že váš kód je pěkný a čistý (a hlavně na první pohled velice srozumitelný, dodává překladatel). Uspořádáním kódu tímto způsobem mohou být všechna volání funkcí programu (ať už používáte vlastní definice nebo voláte funkce z knihovny) napsány v hlavní části vašeho programu vizuálně čistým a organizovaným způsobem. To je dáno tím, že vaše vlastní definice funkcí budou zapsány v programu úplně nejníže, za jeho výkonnou částí. Pozn. př. V některých jiných programovacích jazycích je naopak zvykem zapisovat definice vlastních funkcí před hlavní část programu. Než se dostanete ke čtení jeho hlavní části, musíte se "prokousat" velikým objemem textu s definicemi funkcí.

Jméno _____

Jste na řadě:

Úkol 1: Cvičte definování funkcí

Podívejte se na tento program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 directionToMove=Ed.SPIN_LEFT
13 speedToMoveAt=Ed.SPEED_8
14 distanceToMove=360
15
16 #call the function
17 moveOnClap(directionToMove, speedToMoveAt, distanceToMove)
18
19
20 #user defined functions
21 def moveOnClap(direction, speed, distance):
22     Ed.ReadClapSensor()
23     while Ed.ReadClapSensor()==Ed.CLAP_NOT_DETECTED:
24         pass
25     Ed.Drive(direction, speed, distance)
26
27
```

Řádek 17 v programu je volání funkce, která je definovaná na rádku 21 až 25. Nezapomeňte na naši dohodu, že své vlastní funkce budeme definovat v dolní části programu, aby bylo vše v pořádku a hezky zorganizované.

Napište program a stáhněte ho do Edisona. Spusťte program, abyste zjistili, jak to funguje.

1. Mohli bychom napsat funkci, ve které bude robot Edison jezdit ve čtverci. Dokončete zápis této funkce tím, že zde doplníte chybějící slova:

```
def driveInaSquare():

    for i in range(___):

        Ed.Drive(_____________,_____________,_____________)

        Ed.Drive(_____________,_____________,_____________)
```

2. Napište syntaxi kódu pro volání této funkce. Jinými slovy, co byste ve svém programu muslei napsat pro volání této funkce?

_____

Jméno _____

Napište program, který bude jezdit s Edisonem ve více čtvercích. Budete muset definovat funkci **driveInaSquare** a váš program bude muset tuto funkci volat vícekrát. Stáhněte a spusťte program, abyste zjistili, jak to funguje.

3. Provádí program to, co jste očekávali? Pokud ne, popište, co jste očekávali a co program skutečně udělal. Popište všechny problémy, které jste měli při rozběhu svého programu.

_____

_____

_____

_____

_____

Úkol 2: Navrhněte vlastní funkci

Napište svou vlastní funkci, ve které Edison něco udělá, když tlesknete. Mohli byste nechat Edisona tančit, blikat LEDkou, točit se v kruhu nebo cokoli jiného, co se vám líbí!

Krok 1: Návrh

Nejprve nakreslete vývojový diagram, který graficky shrnuje váš program. Buď vývojový diagram nakreslete na papír nebo použijte program, např. Google Slides (Prezentace Google). Ve vývojovém diagramu používejte správné tvary, které reprezentují start programu, procesy, rozhodovací body a tok programu.

Tvary, které budete potřebovat, závisí na vývojovém diagramu. Minimálně budete potřebovat ovál startu, obdélník procesu a kosočtverec pro rozhodování.

Krok 2: Kód

Převeďte svůj nápad do kódu v aplikaci EdPy. Použijte svůj vývojový diagram jako průvodce a kódujte svou funkci tak, abyste zahrnuli každý krok, který jste naplánovali ve vývojovém diagramu.

Krok 3: Zkouška

Napište testovací program, který obsahuje vaši funkci a další kód, abyste mohli svou funkci 'vykonávat'. (Vykonat funkci znamená zavolat ji ve vašem kódu.) Zjistěte, zda funkce funguje tak, jak jste plánovali, a očekávali, že bude fungovat. Pokud ne, překontolujte vývojový diagram a kód a podle potřeby vše upravte. Pokuste se zjistit, co funguje!

Jméno _____

4. Popište, co vaše funkce dělala.

_____

_____

_____

_____

_____

_____

5. Popište problémy, které jste měli.

_____

_____

_____

_____

_____

Jméno _____

Lekce 7: List aktivity 7.1 – Kalibrace detekce překážek

Citlivost systému detekce překážek Edisona můžete seřídit. Nastavením větší citlivosti systému detekce překážek dokáže Edison odhalit překážky na větší vzdálenost. Naopak, když bude systém méně citlivý, Edison rozpozná jen velmi blízké překážky. Pomocí tohoto listu aktivit nastavte Edisonův systém detekce překážek.

Krok 1: Přečtěte čárový kód

1. Umístěte Edisona na pravou stranu, aby byl otočen směrem k čárovému kódu
2. Třikrát stiskněte kulaté tlačítko záznamu
3. Edison pojede dopředu a naskenuje čárový kód

Čárový kód – Kalibrace detekce překážek

Krok 2: Nastavte maximální citlivost

Po skenování čárového kódu postavte Edisona na stůl a před Edisonem odstraňte všechny překážky. Poté stiskněte tlačítko přehrávání (trojúhelník). Edison je nyní v kalibračním režimu.

Nejprve kalibrujeme citlivost vlevo.

1. Opakovaně stiskněte tlačítko přehrávání (trojúhelník), které zvyšuje citlivost, až se rozbliká levá červená LED.
2. Opakovaně stiskněte tlačítko záznamu (kruhové), které snižuje citlivost, až LED úplně přestane blikat.
3. Stiskněte tlačítko stop (čtvercové) pro přepnutí na kalibraci pravé strany.
4. Opakovaně stiskněte tlačítko přehrávání (trojúhelník), až začne blikat pravá červená LED dioda. Nyní opakovaně stiskněte tlačítko záznamu (kulatý), až LED úplně přestane blikat.
5. Pro dokončení kalibrace stiskněte tlačítko stop.

Zvláštní poznámka: vlastní citlivost

Je také možné nastavit vzdálenost, na kterou jsou překážky zjištěny. Chcete-li to udělat, naskenujte čárový kód "Kalibrace detekce překážek", umístěte překážku před Edisona na vzdálenost, pro kterou chcete, aby Edison detekoval překážky, stiskněte tlačítko přehrávání a opakujte kroky 1 až 5.

Jméno _____

Lekce 7: Pracovní list 7.1 – Detekce překážek infračerveně

V této aktivitě se dozvíte více o infračerveném (IR) světle a o tom, jak Edison používá IR k detekci překážek.

Co je infračervené (IR) světlo?

Existuje široký rozsah světelných vln, z nichž některé jsou lidskému oku viditelné a některé z nich viditelné nejsou. Infračervené světlo, nazývané také IR, není pro lidi viditelné.

> ***Víte že...?*** Infračervený paprsek je druh světla, přestože ho lidé nevidí. Proto bude fungovat i ve tmě. A proto můžete zapnout televizor pomocí dálkového ovládání, i když v místnosti není žádné světlo vidět!

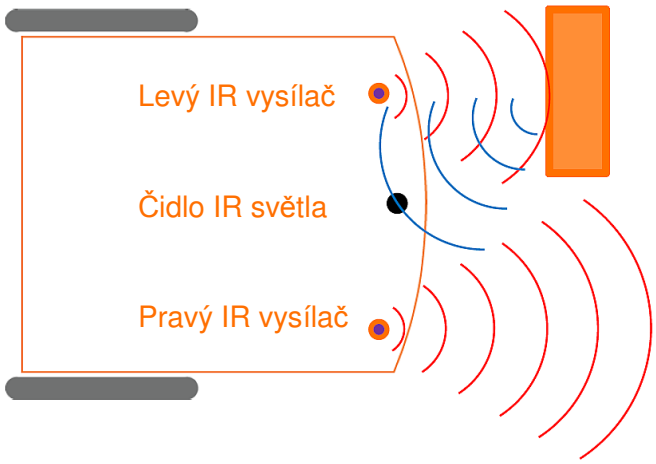
Edison a infračervené světlo

Robot Edison je vybaven infračerveným systémem, který dává robotovi určitý druh "vidění". Tento infračervený systém umožňuje Edisonovi detekovat překážky kolem sebe.

Edisonův infračervený systém je tvořen dvěma diodami na přední straně (někdy je také nazýváme LED diodami). Jedna je vlevo a jedna je vpravo. Edison má na přední straně přímo uprostřed také IR čidlo.

Aby Edison zjistil překážky, infračervené světlo se vydává z levé a pravé diody LED. Pokud se infračervené světlo setká s překážkou, např. se stěnou, odráží se zpět k Edisonovi. IR čidlo Edisona pak detekuje odražené světlo.

Podívejte se na tento obrázek:



Edison vysílá infračervené světlo (je zobrazeno červeně) z levých i pravých IR LED diod. Odražené IR světlo (je zobrazeno modře) se odráží od překážky a je detekováno infračerveným čidlem Edisona.

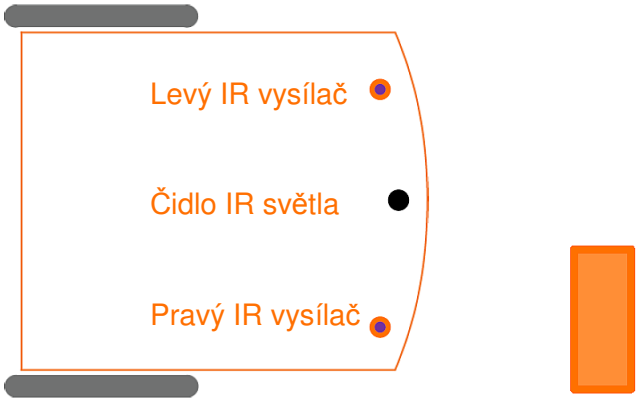
Na obrázku je překážka před Edisonovou levou stranou, ale ne na pravé straně. Proto se odráží pouze IR světlo z levého vysílače.

Z přijatého signálu může Edison určit, že vlevo je překážka, ale vpravo překážka není.

Jméno _____

Jste na řadě:

1. Nakreslete vyzařované IR světlo a odražené IR světlo pro tuto překážku.



Jméno _____

Lekce 7: Pracovní list 7.2 – Zjistěte překážku a zastavte

V této aktivitě budete muset napsat program, aby Edison jel, dokud nenarazí na překážku, a aby se zde zastavil.

Podívejte se na tento program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 Ed.ObstacleDetectionBeam(Ed.ON)
14
15 Ed.Drive(Ed.FORWARD,Ed.SPEED_5,Ed.DISTANCE_UNLIMITED)
16
17 while Ed.ReadObstacleDetection() != Ed.OBSTACLE_AHEAD:
18     pass
19 Ed.Drive(Ed.STOP,1,1)
20
```

Tento program Edisonovi říká, aby jel, dokud nenarazí na překážku.

V tomto programu je několik důležitých věcí.

Podívejte se na řádek 13. Tento řádek "zapíná" Edisonův detekční systém překážek. Kdykoli budete chtít v EdPy programu používat Edisonův paprsek pro detekci překážek, vždy musíte paprsek zapnout ještě předtím, než se paprsek v programu použije.

Nyní se podívejte na řádek 15. Tento řádek nastavuje rychlost jízdy na hodnotu 5. Pokud používáte detekci překážek, musíte použít trochu nižší rychlost, abyrobot mohl detekovat překážku předtím, než se s ní srazí. Je-li rychlost příliš velká, robot na překážku narazí dříve, než ji bude moci detekovat.

Jste na řadě:

Úkol 1: Vyberte překážky

Je potřeba vybrat si ty správné překážky. Je-li překážka příliš malá nebo neodráží dostatek infračerveného světla, Edison ji nemůže rozpoznat. Vyberte objekt, který je neprůhledný, ale není příliš tmavý (např. není černý) a alespoň tak vysoký jako Edison.

Pro tento náš program by byla dobrou překážkou stěna místnosti.

Jméno _____

Úkol 2: Připravte si Edisona

Pokud chcete spustit program používající detekci překážek, je dobré zkontrolovat, zda je detekce překážek robota Edisona kalibrována na požadovanou vzdálenost. Použijte list aktivity 7.1 ke kalibraci Edisona. Možná budete muset kalibrovat svého robota s citlivostí "na míru", abyste se ujistili, že váš robot dokáže detekovat a zastavit u vaší překážky.

Úkol 3: Napište a spusťte program

Napište program pomocí aplikace EdPy a stáhněte jej do Edisona. Poté spusťte program a uvidíte, jak to funguje.

Experimentujte s různými překážkami, abyste zjistili, co Edison dokáže a nedokáže rozpoznat. Můžete také zkusit nastavit kalibraci detekce překážek Edisona na různé vzdálenosti, abyste zjistili, co se pak stane.

1. Z programu odstraňte řádek `Ed.ObstacleDetectionBeam(Ed.ON)`. Zkuste stáhnout a spustit takto upravený program. Co se stalo? Proč se to děje?

_____

_____

_____

2. Přemýšlejte o tom, kde jste už dříve viděli tento typ neviditelné detekce v reálném světě. Popište příklad.

_____

_____

_____

3. Co myslíte, kde jinde by tento typ detekční technologie mohl být použit? Napište alespoň jednu představu o tom, jak tuto technologii můžete použít.

_____

_____

_____

_____

Jméno _____

Lekce 7: Pracovní list 7.3 – Vyhýbání se překážkám

V této aktivitě napište program, aby Edison jel, dokud se nesetká s překážkou, a pak aby se otočil a odjel.

Podívejte se na následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 Ed.ObstacleDetectionBeam(Ed.ON)
14
15 Ed.Drive(Ed.FORWARD,Ed.SPEED_5,Ed.DISTANCE_UNLIMITED)
16
17 while Ed.ReadObstacleDetection() != Ed.OBSTACLE_AHEAD:
18     pass
19 Ed.Drive(Ed.SPIN_RIGHT,Ed.SPEED_5,180)
20 Ed.Drive(Ed.FORWARD,Ed.SPEED_5,10)
21
```

Tento program Edisonovi říká, že má jet dopředu, dokud není zjištěna překážka. Jakmile Edison detekuje překážku, program Edisonovi řekne, ať se otočí o 180 ° a odjede 10 cm daleko.

Jste na řadě:

Úkol 1: Napište a spusťte program

Napište tento program pomocí aplikace EdPy a stáhněte jej do Edisona. Poté spusťte program a uvidíte, jak to funguje.

Po spuštění programu se na program znovu podívejte a přemýšlejte o tom, jak byste jej mohli změnit. Pokuste se upravit kód tak, aby se robot choval jinak, když před sebou zjistí. (*Rada:* Zkuste použít zvuk a světla.)

1. Přemýšlejte o tom, jak můžete zlepšit původní program. Co byste mohli změnit, aby program dělal víc než jen otočení a odjezd po setkání s jednou překážkou?

_____

_____

_____

Jméno _____

Úkol 2: Chyby syntaxe a logické chyby

Programátoři často dělají chyby podobné překlepům, které se nazývají syntaktické chyby. Je důležité, abyste byli schopni vyhledat vlastní chyby syntaxe, abyste mohli kód opravit.

Při programování můžete také udělat jiný typ chyby, nazývaný logická chyba. Při programování je logickou chybou ta chyba, která není syntaktickou chybou.

Pozn. př.: Se syntaktickými chybami nám často a poměrně snadno pomůže překladač. Ale odhalení logické chyby v programu může zabrat celé hodiny práce.

Pokud je v programu logická chyba, kód se nechová tak, jak to programátor očekává. Pokud je v programu EdPy logická chyba, program se obvykle do Edisona normálně stáhne. Když potom program spustíte, nebude se program chovat tak, jak si myslíte, že by to mělo být.

Logická chyba může být naprosto jednoduchá, např. použití nesprávné funkce nebo opominutí nějaké funkce. Příkladem logické chyby by bylo zapomenout zapnout detekční paprsek v programu, který používá detekci překážek.

Podívejte se na následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 Ed.ObstacleDetectionBeam(Ed.ON)
14
15 Ed.Drive(Ed.FORWARD,Ed.SPEED_5, Ed.DISTANCE_UNLIMITED)
16
17 While Ed.ReadObstacleDetection() != Ed.OBSTACLE_NONE
18 pass
19 Ed.Drive(Ed.SPIN_RIGHT,Ed.SPEED_5,145)
20 Ed.Drive(Ed.FORWARD, Ed.SPEED_5,20)
21
```

Program je míněn tak, aby Edison jel vpřed, dokud nenarazí na překážku, pak se otočí o 135° a odjede od překážky 20 cm.

V programu je ale pět chyb. Najdete všechn pět chyb programu a poznáte, zda jsou to syntaktické chyby nebo logické chyby? Vyplňte své odpovědi v tabulce na další stránce.

Rada: Tento program můžete napsat v EdPy a tlačítko **Zkontroluj program** (Check code) vám pomůže najít syntaktické chyby.

Jméno _____

Č. chyby	Č. řádku	Druh chyby (syntaktická n. logická)	Popis chyby
1			
2			
3			
4			
5			

Jméno _____

Lekce 7: Pracovní list 7.4 – Zjistěte překážku jako událost

V této aktivitě napíšete program řízený událostí, díky němuž Edison jede nepřetržitě dopředu a zároveň se vyhne překážkám.

Programování řízené událostmi

Při programování je událost něco, co se vyskytuje mimo programový kód, je to z reálného světa, ale ovlivňuje to běh programu. Událostí může být stisknuté tlačítko nebo informace přenášená z čidla.

Mnoho programovacích jazyků včetně Pythonu umožňuje programátorům vytvářet kód, který může reagovat na určitou skupinu událostí. Tento typ programování se nazývá programování řízené událostmi.

Kdykoli napíšete program, který je řízen událostí, budete také muset napsat blok kódu, který tyto události zpracovává. Existují dva hlavní způsoby, jak reakci na událost provést – buď tím, že program čeká na událost ve smyčce (a hlavní program v tuto chvíli nemůže efektivně běžet, protože procesor je obsazen tím čekáním, stále znova a znova testuje nějakou stejnou podmínku, která se ale tak často nezmění), nebo se čeká pomocí přerušení – a to si teď velmi podrobně vysvětlíme.

Tuto lekci přerušujeme a mluvíme o ... přerušení :)

Jak už víte, programy se obvykle pohybují kódem postupně řádek po řádku. Existují způsoby, jak umožnit, aby se kód pohyboval jiným způsobem, např. pomocí cyklů.

Můžete také ovlivnit způsob běhu programu pomocí 'přerušení' (**interrupt**).

Přerušení je část kódu, která na chvíli pozastaví hlavní program a místo něj se spustí sama. Jakmile je program zpracovávající přerušení ukončen, běh programu se vrátí tam, kde se v hlavním programu v okamžiku výskytu přerušení přestalo pracovat (kde se z hlavního programu na chvíli odskočilo).

Přerušení jsou vždy funkce definované někde v programu. Jedná se obecně o krátké úseky kódu, které jsou určeny k provedení určitého úkolu, aniž by došlo k významnějšímu narušení toku hlavního programu.

Programátoři používají přerušení, protože přerušení umožňují programu zareagovat na událost kdykoli během běhu hlavního programu. Jinými slovy, použitím přerušení nemusí programátor předvídat přesně, kdy během programu tato událost nastane. Kdyby program chtěl na událost reagoval běžně, tedy sekvenčně, bylo by obtížné umísťovat dílčí kód pro událost někam dovnitř hlavního programu. Prakticky jistě bychom se do správného okamžiku netrefili! Po oddělení obou programů máme někde v paměti hlavní program a v JINÉ části paměti máme všechny potřebné obslužné programy a funkce. (Všechny říkáme proto, že událostí je v reálném světě velmi mnoho druhů, a tím spíš se tu ukazuje výhoda, která vzniká oddělením programů pro události od hlavního programu.)

Jméno _____

Obslužné programy událostí a přerušení

Při použití přerušení v programování řízeném událostmi budete muset použít obslužný program události. Obslužný program události je způsob, jak navázat dílčí kód na určitou událost.

Pozn. př.: Anglicky je obslužný program události **event handler** *a názorněji plným významem 'obslužný program přerušení způsobeného událostí', v učebnicích najdeme třeba také sousloví 'obsluha události', 'zpracování události' aj. – vše toto jsou synonyma nebo zkrácená vyjádření. A následující dva odstavce překladatel trochu doplnil, aby studenti získali o přerušení ucelenou představu hned tady.*

Chceme-li použít obslužný program události, musíme nejprve napsat a hlavně nastavit neboli 'zaregistrovat' tento **NÁŠ** obslužný program události v hlavním programu. Když je obslužný program události (**event handler**) registrován, tak se v této fázi pomocí registrace události (vnitřně nějak technicky) připraví sledování té dané události (a obvykle se dále kolem události ***už nic nedělá***, a tak hlavní program má nyní volný prostor k jiné nerušené práci).

Když kdykoli později k této 'obsluhované' události dojde, spustí se přerušení a tím zapracuje výkonná část obslužného program události, která mimo jiné pozastaví běh hlavního programu *(=**přeruší ho**, proto se tomu všemu okolo říká **přerušení**)* a zavolá (spustí) příslušnou **NAŠI funkci** *(a jak jsme si řekli před chvílí – obsluha přerušení je v Pythonu **navenek vždy funkce** – a je to proto tak krásně jednoduché na používání)* a po vykonání funkce (po splnění úkolu) se řízení vrátí do hlavního programu, do toho místa, kde se předtím přestalo pracovat. Hlavní program tak vlastně ani neví (a neumí to zpravidla ani zjistit), že byl na chvíli přerušen!

Pozn. př.: Současně registrovaných událostí může být a taky v praxi bývá velmi mnoho (desítky, stovky a i více) – ale protože se při čekání na tyto události nespotřebovává žádný výkon procesoru, je přerušení jeden z nejefektivnějších způsobů, jak psát programy pro svět skutečných věcí.

Podívejte se na tento program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 Ed.ObstacleDetectionBeam(Ed.ON)
13 Ed.RegisterEventHandler(Ed.EVENT_OBSTACLE_AHEAD, "avoidObstacle")
14
15 while True:
16     Ed.Drive(Ed.FORWARD,Ed.SPEED_5,Ed.DISTANCE_UNLIMITED)
17
18 def avoidObstacle():
19     Ed.Drive(Ed.SPIN_RIGHT,Ed.SPEED_5,180)
20     Ed.ReadObstacleDetection()
21
```

V nástroji EdPy používáme pro registraci obsluhy události funkci `Ed.RegisterEventHandler`.

Jméno _____

Funkce `Ed.RegisterEventHandler` má dva parametry. Prvním parametrem je událost, která se má vyskytnout (na kterou se má zareagovat), a druhý parametr je (název) funkce, která bude volána vždy, když tato událost nastane.

Podívejte se na řádek 13 programu. Tento řádek registruje obslužný program událostí tak, že kdykoli nastane `EVENT_OBSTACLE_AHEAD`, bude volána funkce `avoidObstacle`.

Jste na řadě:

Napište program pomocí aplikace EdPy a stáhněte jej do Edisona. Poté spusťte program a uvidíte, jak to funguje.

1. Popište, co robot dělá.

_____

_____

_____

2. Když robot detekuje vepředu překážku, které řádky programu se provedou? Proč program spouští tyto řádky?

_____

_____

_____

_____

3. Proč je `Ed.ReadObstacleDetection` součástí tohoto programu? Jinými slovy, co dělá řádek 20? *Rada:* Pro nápovědu se podívejte zpět na pracovní list 2.5, nebo zkuste řádek 20 odstranit a spustit program.

_____

_____

_____

_____

Zkuste to!

Co jiného by ještě mohl Edison udělat, když zjistí, že před ním stojí nějaká překážka? Pokuste se modifikovat kód tak, aby se Edison při zjištění překážky vpředu choval jinak. Experimentujte s programem, abyste zjistili, co funguje a co ne.

Jméno _____

Lekce 7: Pracovní list 7.5 – Detekce překážek vpravo a vlevo

V této činnosti napíšete program pro Edisona, který má reagovat na překážky vlevo nebo vpravo od robota. K tomu použijeme příkaz **if** *(pozn. př.: přeloženo do ČJ "příkaz KDYŽ", ale zde jsme se dohodli, že pro klíčová slova Pythonu budeme používat originální slova v AJ)*.

Příkazy if

Důležitou součástí programu je rozhodování. Nejběžnějším způsobem, jak to udělat, je použít příkaz **if**.

Příkaz **if** se ptá, zda je podmínka pravdivá nebo nepravdivá. Pokud je výsledek **true**, pak program provede blok příkazů (řádků) v příkazu **if**. Pokud je výsledek **false**, program ignoruje příkazy v příkazu **if** a přesune se na další řádek kódu mimo příkaz **if**. (Tj. přejde za příkaz **if** na další už stejným způsobem neodsazený řádek nebo na konec programu, pokud už žádné další řádky v hlavním programu nejsou.)

Podívejte se na následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 Ed.ObstacleDetectionBeam(Ed.ON)
13
14 while True:
15     if Ed.ReadObstacleDetection() != Ed.OBSTACLE_NONE:
16         Ed.PlayBeep()
17         Ed.ReadObstacleDetection()
18
```

Tento program používá příkaz **if**, který umožňuje robotovi rozhodovat bez lidského vedení. Když se robot může sám o sobě rozhodnout tímto způsobem, nazýváme ho autonomním robotem.

Jste na řadě:

Úkol 1: Pípnutí, pokud se zjistí překážka

Napište výše uvedený program pomocí aplikace EdPy, stáhněte ho do Edisona a spusťte program. Pak zkuste přemístit překážku (například vaši ruku) dovnitř a ven z Edisonova detekčního paprsku překážek, abyste zjistili, co se stane.

Jméno _____

1. Edison nyní může reagovat odlišně na různé podněty a činit 'rozhodnutí', co dělat. Znamená to, že Edison má inteligenci? *Rada:* Možná můžete trochu prozkoumat umělou inteligenci, což vám pomůže rozhodnout.

_____

_____

_____

_____

_____

Úkol 2: Pípnutí, pokud se zjistí překážka, jinak se zapni světlo

Kromě toho, že programu říkáte, co má dělat, když je podmínka příkazu **if** pravdivá, můžete programu také říct, co dělat, když tato podmínka pravdivá není.

Příkaz if, else

Používání příkazu **if** spolu s **else** umožňuje psát programy vykonávající složitější rozhodnutí. Příkaz **if**/**else** je v podstatě způsob, jak rozhodnout mezi dvěma věcmi. (Samotný příkaz **if** z předchozího úseku totiž rozhoduje mezi jedinou věcí a jejím "neuskutečněním".)

Sledujte Billa Gatese, zakladatele společnosti Microsoft, jak vysvětluje příkazy **if**/**else** a rozhodování v programování: https://youtu.be/fVUL-vzrlcM

V Pythonu je syntaxe příkazu **if**/**else** tato:

```
if expression:
    statement(s)
else:
    statement(s)
```

Pozn. př.: `expression` je nám už dříve známý **výraz** a `statement` je **příkaz**.

Program se posouvá postupně shora dolů, počínaje podmínkou `if`. Pokud podmínka příkazu `if` má hodnotu **true**, program spustí odsazený kód v sekci za `if` a přeskočí sekci za `else`. Má-li však podmínka příkazu `if` hodnotu **false**, program přeskočí tuto první část odsazeného kódu a místo toho spustí odsazený kód za `else`.

Jméno _____

Napište tento program:

```
1
2  #------------Setup---------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 Ed.ObstacleDetectionBeam(Ed.ON)
13
14 while True:
15     if Ed.ReadObstacleDetection() != Ed.OBSTACLE_NONE:
16         Ed.LeftLed(Ed.OFF)
17         Ed.PlayBeep()
18         Ed.TimeWait(100, Ed.TIME_MILLISECONDS)
19         Ed.ReadObstacleDetection()
20     else:
21         Ed.LeftLed(Ed.ON)
22
```

Stáhněte a spusťte program. Zkuste přemístit překážku dovnitř do cesty paprsku detekce překážek Edisona a zase mimo paprsek, abyste zjistili, co se stane.

Tento program má dvě cesty: jednu při zjištění překážky a druhou, když není zjištěna překážka.

Příkaz if, elif, else

Můžete také vytvořit program, který rozhoduje za použití více než dvou podmínek. K tomu použijete jinou strukturu syntaxe Pythonu:

```
if expression:
    statement(s)
elif expression:
    statement(s)
else:
    statement(s)
```

`elif` je zkrácené slovo Pythonu pro **`else if`** – takto se to obvykle píše v jiných programovacích jazycích. Strukturu s `elif` můžete použít k napsání programu s více podmínkami.

Program, který používá strukturu příkazu **if – elif – else** se také pohybuje postupně odshora dolů. Jakmile však bude splněna některá z více podmínek '`expression`', program spustí příslušný blok odsazeného kódu. A dále, když už se dle splněné podmínky jedou spustil jakýkoli tento odsazený kód uvnitř kterékoli části struktury příkazu **if**, kompletní zbytek struktury se přeskočí a program se přesune na další řádek kódu mimo strukturu.

To znamená, že pokud podmínka za `if` zcela nahoře je **true**, program spustí odsazený kód pro výraz `if` a přeskočí oddíly `elif` a sekci `else`, pokud ta existuje. Pokud podmínka za `if` zcela nahoře je **false**, program přeskočí tuto část odsazeného kódu a přesune se do nejbližší

další části, tedy na první `elif` a pokračuje ve vyhodnocování podmínek. Pokud je tato první podmínka `elif` **true**, program spustí její odsazený kód a přeskočí vše pod ním ve struktuře příkazu (tedy libovolné jiné `elif` a podmínku `else`, pokud existuje). Pokud je tato podmínka `elif` **false**, program se přesune na další část struktury příkazu a vyhodnocuje se dále atd.

Pozn. př.: Často se věty zkracují takto: "pokud je `if` **true**" znamená přesněji "pokud je logický **výraz** za klíčovým slovem `if` pomíněného **příkazu if** pravdivý (true)".
Když mluvíme o **příkazu if** jako celku, myslíme zpravidla kteroukoliv z jeho variant:

`if`
`if  else`
`if  elif  else`
	`if  elif  elif ... elif`
	`if  elif  elif ... elif  else`

Úkol 3: Detekuj překážku nalevo nebo napravo

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 Ed.ObstacleDetectionBeam(Ed.ON) #turn on obstacle detection
13
14 while True:
15     Ed.Drive(Ed.FORWARD, Ed.SPEED_1, Ed.DISTANCE_UNLIMITED)
16
17     obstacle=Ed.ReadObstacleDetection()
18
19     if obstacle>Ed.OBSTACLE_NONE: #there is an obstacle
20         Ed.Drive(Ed.BACKWARD, Ed.SPEED_5, 7)
21         if obstacle==Ed.OBSTACLE_LEFT:
22             Ed.Drive(Ed.SPIN_RIGHT, Ed.SPEED_5, 90)
23         elif obstacle==Ed.OBSTACLE_RIGHT:
24             Ed.Drive(Ed.SPIN_LEFT, Ed.SPEED_5, 90)
25         elif obstacle==Ed.OBSTACLE_AHEAD:
26             Ed.Drive(Ed.SPIN_RIGHT, Ed.SPEED_5, 180)
27         Ed.ReadObstacleDetection() #clear any unwanted detections
28
```

Tento program má tři různé cesty (a nemá **else**), po kterých může jít, když je zjištěna překážka – a to v závislosti na tom, kde se nachází překážka ve vztahu k Edisonovi.

Napište tento program pomocí aplikace EdPy a stáhněte jej do Edisona. Program spusťte a uvidíte, jak to funguje. *Všimněte si dvojtečky za **každým** `expression`.*

2. Když spustíte tento program, vysvětlete vlastními slovy, co robot dělá, když

je detekována překážka vpředu: _____

je detekována překážka vpravo: _____

je detekována překážka vlevo: _____

Jméno _____

Lekce 8: Pracovní list 8.1 – Čidlo pro sledování vodící čáry

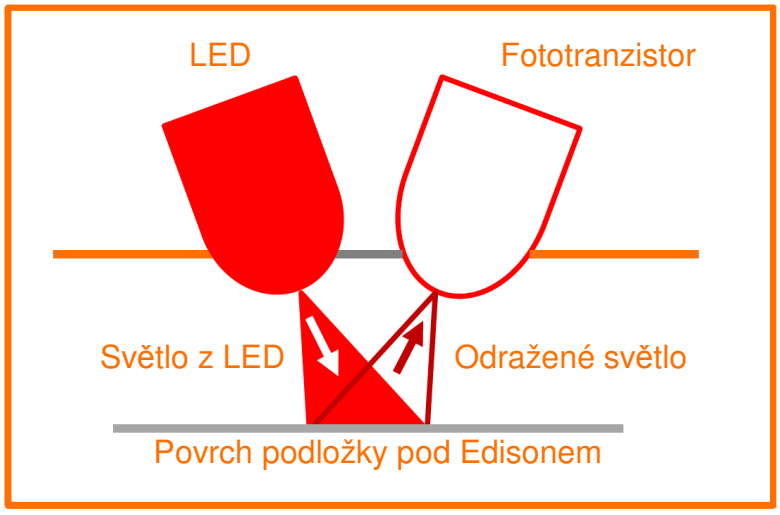
V této aktivitě se dozvíte o čidle pro sledování čáry robota Edisona a o tom, jak může Edison toto čidlo použít, aby zjistil, zda je na odrazném nebo nereflexním povrchu.

Jak funguje Edisonovo čidlo pro sledování čáry?

Robot Edison je vybaven čidlem pro sledování čáry umístěným v blízkosti spínače napájení na spodní straně robota. Tento snímač je tvořen dvěma hlavními elektronickými součástmi:

1. LED dioda s červeným světlem a
2. fototranzistor (světelné čidlo).

Na obrázku je průřez Edisonovým čidlem pro sledování čáry:



LED dioda čidla čáry svítí na povrch, po kterém robot Edison jezdí.

Fototranzistor snímá světlo. Fototranzistor měří množství světla, které se odráží zpět od povrchu pod Edisonem.

Jste na řadě:

Úkol 1: Černá nebo bílá

Když se do Edisonova fototranzistoru odráží více světla, poskytuje vyšší údaj o světle.

Pokuste se zjistit, zda odráží více světla bílá nebo černá plocha.

Použijte list aktivity 8.1 nebo jeden kus bílého a jeden kus černého papíru. Zapněte Edisona a dvakrát stiskněte kulaté tlačítko, aby se rozsvítila LED dioda pro sledování čáry. Zvedněte Edisona kousek nad papír a pozorně se podívejte na kulatou světelnou skvrnu, kterou vytváří na povrchu LED. Porovnejte, jak je jasné světlo na černém povrchu a pak na bílém povrchu.

1. Je světelná skvrna vržená LED jasnější, když se nachází na černém nebo na bílém povrchu?

_____

Jméno _____

Úkol 2: Červená, zelená a modrá

Jak jste viděli v úkolu 1, od bílého povrchu se odráží více světla než z černého povrchu. Proto se na bílém povrchu projeví světlo jako jasnější.

Když je fototranzistor na bílém povrchu, poskytuje vyšší údaj o světle, než když je na černém povrchu. Černý povrch je proto považován za "nereflexní" a bílý povrch je považován za "reflexní" / "odrazivý".

Schopnost fototranzistoru určit, zda je Edison na reflexním nebo nereflexním povrchu umožňuje naprogramovat robota tak, aby reagoval na povrch, po němž se pohybuje.

2. Přemýšlejte o tom, jak by čidlo čáry reagovalo na následující barvy povrchu. Mohlo by čidlo vnímat konkrétní barvu jako reflexní nebo nereflexní? Nezapomeňte, že LED dioda Edisonova čidla čáry emituje červené světlo.
(*Rada:* Můžete své odpovědi otestovat pomocí listu aktivity 8.1.)

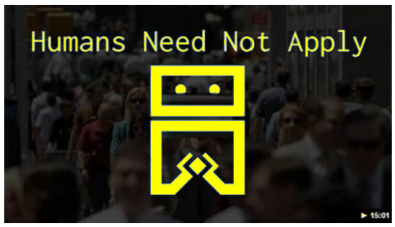
Červený povrch _____

Zelený povrch _____

Modrý povrch _____

Video – Lidé už nemusí být potřeba!

Sledování čáry je úplně základní robotické chování, které se dnes používá v mnoha automatech a továrnách. Bude to stejné v budoucnu? Jak bude vypadat budoucnost s mnoha roboty? Nastane to někdy?



Podívejte se na video ***Humans Need Not Apply***, abyste se dozvěděli více o tom, co by široké využívání robotů mohlo znamenat pro budoucnost lidí v práci: https://www.youtube.com/watch?v=7Pq-S557XQU (15 minut, anglicky)

Jméno _____

Lekce 8: Pracovní list 8.2 – Jeď až k černé čáře

V této aktivitě napíšete program tak, aby Edison jel vpřed po bílém (reflexním) povrchu, dokud nepřejede černou (nereflexní) čáru.

Podívejte se na tento program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 Ed.LineTrackerLed(Ed.ON)
14
15 Ed.Drive(Ed.FORWARD,Ed.SPEED_6,Ed.DISTANCE_UNLIMITED)
16
17 while True:
18     if Ed.ReadLineState() == Ed.LINE_ON_BLACK:
19         Ed.PlayBeep()
20         Ed.Drive(Ed.STOP,Ed.SPEED_6,0)
```

Podívejte se na řádek 13. Tento řádek volá funkci **`Ed.LineTrackerLed()`**. a změní stav LED na zapnuto ('on').

Stejně jako u Edisonova paprsku pro detekci překážek musíte před použitím čidla pro sledování čáry v programu nejprve čidlo zapnout. Zapnutím čidla sledování čáry se aktivuje také červená LED dioda sledovače.

Nyní se podívejte na řádek 19. Tento řádek volá funkci **`Ed.PlayBeep()`**. Tato řádek nemá vliv na způsob, jakým program pro sledování čáry funguje, ale účelem tohoto řádku je ladění.

Ladění

Ladění je proces hledání chyb neboli 'much' v programu. Aby se dal sledovat průběh programu, tak programátoři často vkládají do programu řádky, jako je zde řádek 19.

Řekněme, že tento program spustíte, ale robot se nezastaví na černé čáře. Existují dva možné důvody: (1) robot nemusí detekovat černou čáru nebo (2) může dojít k chybě v závěrečném příkazu **`Ed.Drive()`**,

Pokud uslyšíme pípnutí, víme, že byla zjištěna černá čára. Proto víme, že chyba byla v dalším příkazu. Tento přidaný ladicí kód nám pomáhá snadněji určit chybu.

Pro ladění mohou být také použity jiné funkce, např. příkaz **`Ed.LeftLed()`**. Tento příkaz byste mohli použít pro zapnutí levé LED indikující, že program dosáhl určitý bod.

Jméno _____

Jste na řadě:

Pomocí aplikace EdPy napište program a stáhněte je do Edisona. Pro testování programu použijte černou čáru na listu aktivity 8.1. Můžete také nakreslit černou čáru na kus bílého papíru nebo použít černou elektrikářskou pásku na bílém stole.

> **Pozn.:** Kdykoli používáte v programu Edisonovo čidlo pro sledování čáry, vždy odstartujte s robotem na bílé (reflexní) ploše - nikdy na černém (nereflexním) povrchu.

Dejte Edisona na bílý povrch a rozjeďte robota směrem k černé čáře.

Potom zkuste spustit program znovu s použitím každé ze tří barevných čar na listu aktivity 8.1, zkuste jednu po druhé. Rozjeďte Edisona směrem k určité barevné čáře, abyste otestovali, zda robot čáru detekuje a zastaví.

1. Existují barvy, které Edison nedokáže dobře rozpoznat? Pokud ano, které to jsou)?

   _____

2. Proč myslíte, že jste dostali odpověď, kterou jste napsali u otázky č, 1? Proč nemůže Edison tuto barvu zjistit?

_____

_____

_____

3. Představte, že programujete svého Edisona, aby jel slalomovou dráhu se třemi slalomovými praporky. Popište, jak byste mohli v programu použít funkce `Ed.PlayBeep()`, `Ed.LeftLed()` nebo `Ed.RightLed()` pro účely ladění a co by udělaly.

_____

_____

_____

_____

_____

_____

_____

Jméno _____

Lekce 8: Pracovní list 8.3 – Jízda uvnitř hranic

V této aktivitě je třeba napsat program, který bude Edisona udržovat uvnitř černého rámečku pomocí čidla pro sledování čáry.

Nejprve musíme program naplánovat pomocí pseudokódu.

Pseudo kód

Plánování programu před zahájením kódování je v programování důležitá a užitečná dovednost.

Jeden ze způsobů, jak to udělat, je pomocí vývojového diagramu, který stručně znázorňuje tok programu, o tom jste se dozvěděli v lekci 6. Pseudokód je další způsob, jak program znázornit ještě předtím, než začnete kódovat.

Pseudokód je způsob, jak napsat program zjednodušeným a snadno čitelným způsobem. Pseudokód připomíná nějaký zjednodušený programovací jazyk, ale není založen na žádném konkrétním programovacím jazyce, takže je bez syntaxe. Místo toho pseudokód používá angličtinu pro popis toho, co bude program dělat. Proto se pseudokód také nazývá 'strukturovaná angličtina'.

Když plánujete svůj program pomocí pseudokódu, měli byste odsazovat řádky stejným způsobem jako v programovacím jazyce, aby byl pseudokód snadno čitelný a srozumitelný.

Zde je příklad pseudokódu, který popisuje program, kdy Edison zůstává v nereflexní hranici pomocí čidla pro sledování čáry:

```
Turn the line tracker on
Loop forever
    Drive Forward
    If the robot detects a black line then
        Stop
        Play a beep
        Spin right 135°
```

Jako překladatel se zde pokusím pseudokód přeložit do 'strukturované češtiny', i když vím, že angličtina k programování odjakživa 'natvrdo' patří a budoucímu programátorovi se dobrá angličtina velice vyplatí.

```
Zapni čidlo čáry
Dělej nekonečnou smyčku
    Jeď vpřed
    Když robot zaznamenal černou čáru tak
        Zastav
        Zapípej
        Otoč se doprava o 135°
```

Jméno _____

Zde je odpovídající program v EdPy

```
11  #--------Your code below-----------
12
13  Ed.LineTrackerLed(Ed.ON)
14
15  while True:
16      Ed.Drive(Ed.FORWARD, Ed.SPEED_3, Ed.DISTANCE_UNLIMITED)
17      if Ed.ReadLineState() == Ed.LINE_ON_BLACK:
18          Ed.Drive(Ed.STOP, Ed.SPEED_3, Ed.DISTANCE_UNLIMITED)
19          Ed.PlayBeep()
20          Ed.Drive(Ed.SPIN_RIGHT, Ed.SPEED_5, 135)
21
```

Porovnejte pseudokód s programem. Vidíte, jak se pseudokód přeložil do programu v Pythonu?

Jste na řadě:

Napište tento program:

```
1
2   #-------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12
13  Ed.LineTrackerLed(Ed.ON)
14
15  while True:
16      Ed.Drive(Ed.FORWARD, Ed.SPEED_3, Ed.DISTANCE_UNLIMITED)
17      if Ed.ReadLineState() == Ed.LINE_ON_BLACK:
18          Ed.Drive(Ed.STOP, Ed.SPEED_3, Ed.DISTANCE_UNLIMITED)
19          Ed.PlayBeep()
20          Ed.Drive(Ed.SPIN_RIGHT, Ed.SPEED_5, 135)
21
```

Stáhněte program do Edisona a spusťte, abyste viděli, jak pracuje.

Použijte list aktivity 8.2 jako hranici pro otestování programu. Můžete také vytvořit vlastní ohraničení pomocí velkého kusu papíru a tlustého černého fixu, nebo pro vytvoření velké oblasti (hranice) můžete použít černou elektrikářskou pásku na bílém stole nebo na podlaze.

Upravte program:

Zkuste odstranit zastavení odstraněním řádku 18 programu. Pak experimentujte se změnou rychlosti jízdy. Otestujte své upravené programy, abyste zjistili, co se stane.

Jméno _____

1. Jak rychle může robot jezdit, než se objeví nějaké problémy?

_____

2. Co se stane, když robot jede příliš rychle?

_____

_____

_____

Jméno _____

Lekce 8: Pracovní list 8.4 – Sledujte čáru

V této aktivitě použijete Edisonovo čidlo pro sledování čáry a napíšete program v EdPy, který přiměje Edisona sledovat černou čáru.

K tomu je třeba nejdříve vytvořit algoritmus pro sledování černé čáry.

Co je to algoritmus?

Řekněme, že chcete naučit své přátele, jak vyrábět ovocné koláče. Pokud víte, že všichni vaši přátelé mají jablka, stačí napsat jeden recept na jablečný koláč.

Ne všichni vaši přátelé však nutně musí mít jablka. Co když jeden z nich má borůvky, jiný má třešně a třetí má jablka? Proto nemohou všichni pracovat podle receptu na jablečný koláč. Budete muset napsat samostatný recept pro každý druh ovoce.

A co když nevíte, jaké ovoce mají vaši přátelé? Jak byste je mohli naučit, jak vyrábět ovocné koláče? Bez ohledu na to, jaké ovoce mají, všichni vaši přátelé musí dodržovat stejné základní pokyny: připravit si těsto, naplnit koláč ovocem a poté koláč upéct.

Tato nová sada pokynů je příkladem algoritmu.

Algoritmus je rozsáhlá sada pokynů k vyřešení problémů z nějaké množiny. Algoritmus určuje proces nebo soubor pravidel, které je třeba dodržovat, aby se vyřešil jakýkoli problém z této množiny problémů.

Algoritmy v programování

Algoritmy v programování

V programování máme často skupiny (množiny) problémů, které chceme vyřešit.

V této činnosti např. chceme, aby Edison sledoval jakoukoliv černou čáru. Naše množina problémů je tedy "sledovat každou černou čáru". Jakákoli konkrétní čára, kterou pro Edisona uděláte, je nový problém uvnitř této množiny.

Řekněme, že nakreslíte Edisonovi černou čáru, kterou bude sledovat. Chcete-li, aby Edison sledoval vaši čáru, můžete napsat program, který Edisona řídí přesnou cestou po čáře. Pokud vytvoříte novou čáru, budete muset pro tuto novou čáru napsat celý nový program.

Místo toho však můžete vytvořit algoritmus.

Algoritmus vytváří program, který bude pracovat na všech problémech z množiny. Tím není nutno napsat pro každý nový problém zcela nový program.

Abychom vyřešili naši množinu problémů, potřebujeme algoritmus k vytvoření sady instrukcí, které budou fungovat pro jakoukoliv černou čáru.

Můžete naplánovat algoritmus pomocí pseudokódu nebo vývojového diagramu, stejně jako to děláte, když plánujete program.

Jméno _____

Zde je algoritmus v pseudokódu, který umožní Edisonovi sledovat libovolnou černou čáru:

```
Turn the line tracker on
Loop forever
    If the robot detects a black line then
        Drive Forward Right
    Else
        Drive Forward Left
```

```
Zapni čidlo čáry
Dělej pořád smyčku
     Když robot vidí černou čáru tak
         Jede dopředu doprava
     Jinak
         Jede dopředu doleva
```

Tento algoritmus říká, že pokud se čidlo sledování čáry machází na nereflexní ploše (černé), potom by měl robot Edison jet dopředu doprava a tak odjet s čidlem směrem k bílému povrchu. Jiank, když snímač sledování čáry na černém povrchu není, pak by měl robot jeto dopředu doleva a odjet s čidlem směrem k černému povrchu. Tímto způsobem se robot plynule pohybuje dopředu a sleduje hranu čáry.

Všimněte si, že v pseudokódu nejsou uvedeny žádné rychlosti nebo vzdálenosti. Tyto detaily jsou většinou ponechány na fázi kódování.

Jste na řadě:

Přeložte algoritmus pseudokódu do programu v Pythonu, aby Edison sledoval černou čáru. Experimentujte s různými rychlostmi a vzdálenostmi, abyste dosáhli co nejhladšího sledování čáry. Stáhněte si kód do Edisona a vyzkoušejte jej pomocí dráhy na listu aktivity 8.2.

1. Jaká byla nejlepší nalezená kombinace rychlosti a vzdálenosti pro dosažení hladkého sledování čáry?

_____

2. Jak vypadá váš program v Pythonu? Sem napište svůj kód.

_____

_____

_____

_____

Jméno _____

_____

_____

_____

_____

_____

_____

_____

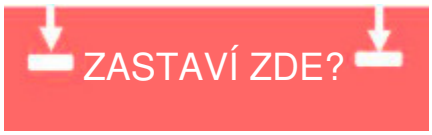
Zkuste to!

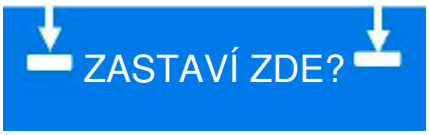
Vytvořte na bílém papíru vlastní čáru černým fixem nebo černou páskou. Dokáže Edison sledovat vaši čáru?

_____

Jméno _____

Lekce 8: List aktivity 8.1

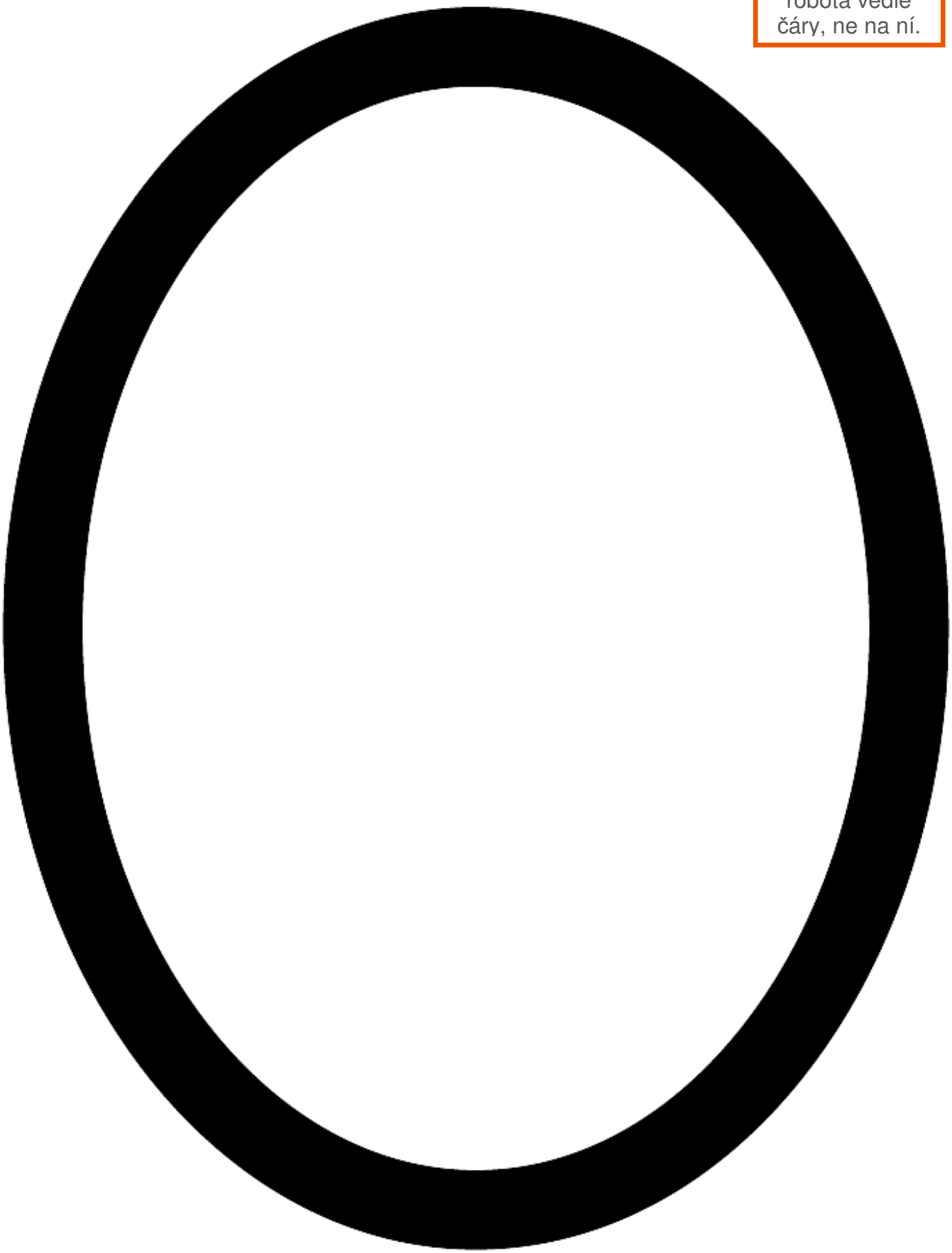




Jméno _____

Lekce 8: List aktivity 8.2

Pamatujte si!
Vždy spusťte robota vedle čáry, ne na ní.



Jméno _____

Lekce 9: Pracovní list 9.1 – Světelný alarm

V této aktivitě naprogramujte Edisona, aby zvonil na poplach při zapnutí světel v místnosti.

Použití Edisonových světelných senzorů v programech

Robot Edison má dvě světelná čidla, jedno vlevo a druhé vpravo, která dokážou rozpoznat **viditelné** světlo. Tato čidla lze použít k naprogramování Edisonovy reakce na světlo.

V programu můžeme těmito čidly přečíst množství detekovaného světla a toto změřené množství vrátit jako digitální hodnotu.

Podívejte se na následující program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 while Ed.ReadLeftLightLevel() < 100:
14     pass
15 while True:
16     Ed.PlayBeep()
17
```

Program používá funkci **`Ed.ReadLeftLightLevel()`** na řádku 13. Tato funkce přečte zpět do programu úroveň světla z levého čidla světla Edisona a vrátí hodnotu mezi 0 a 1023.

Protože funkce **`Ed.ReadLeftLightLevel()`** vrací hodnotu, můžeme s touto hodnotou matematicky počítat. [Pozn. př.: Můžeme ji použít v nám již známém výrazu pro porovnání.] První cyklus v programu používá výraz k určení toho, co má dělat. V prvním cyklu se říká, že uděláme `pass` (tj. vynecháme toto kolo cyklu, jinými slovy se neudělá nic) tehdy, když hodnota vrácená funkcí **`Ed.ReadLeftLightLevel()`** je menší než sto (<100).

Když je vrácená hodnota větší než sto (>100), podmínka za **while** se stane **false** (nepravda) a program opustí první cyklus a přejde do dalšího cyklu, který nepřetržitě vydává alarm.

Jste na řadě:

Napište program a stáhněte ho do Edisona. Nejprve umístěte Edisona do tmy nebo zakryjte čidlo levého světla a stisknete tlačítko přehrávání.

Jakmile rozsvítíte světla nebo odkryjete levé čidlo světla, tak Edison spustí poplach.

Jméno _____

1. Přemýšlejte o situaci v reálném životě, kdy by tento typ alarmu byl užitečný. Popište situaci a způsob, jakým může být alarm použit.

_____

_____

_____

_____

2. Jaké změny je nutné v programu provést, aby alarm reagoval naopak na tmu?

_____

_____

_____

_____

Pozn. př.: Uměli byste program upravit tak, aby alarm spustil řekněme až za 10 vteřin po jeho aktivaci tlačítkem přehrávání? Reálné alarmy se takto často chovají.

_____

_____

_____

_____

Jméno _____

Lekce 9: Pracovní list 9.2 – Automatické osvětlení

V této aktivitě napíšete program, kdy Edison zapne dvě LED světla, když nastane tma.

Podívejte se na tento program:

```
1
2  #-------------Setup---------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 Ed.Drive(Ed.FORWARD, Ed.SPEED_5, Ed.DISTANCE_UNLIMITED)
13
14 while True:
15     if Ed.ReadLeftLightLevel() < 100:
16         activateBothLights(Ed.ON)
17     else:
18         activateBothLights(Ed.OFF)
19
20
21 def activateBothLights(stateOfLed):
22     Ed.LeftLed(stateOfLed)
23     Ed.RightLed(stateOfLed)
24
```

V tomto programu používáme běžný symbol 'menší než' (<) pro určení větve, kterou program provede (kterou program projde).

Pokud je hodnota vrácená z funkce **`Ed.ReadLeftLightLevel()`** menší než 100, aktivuje se větev s funkcí **`activateBothLights()`** s parametrem vstupu **`Ed.ON`**. (Pozn. př.: Tato funkce **`activateBothLights()`** je naše vlastní, námi definovaná, není z knihovny **`Ed`**, proto taky před ni nepíšeme (a ani ***nesmíme*** psát) **`Ed.`** Funkce je uživatelsky definována v našem programu úplně dole, pomocí klíčového slova `def`.

V opačném případě, když hodnota vrácená funkcí **`Ed.ReadLeftLightLevel()`** je větší než 100, program se přesune do části `else` příkazu `if`, který také volá naši funkci **`activateBothLights()`**, ale nyní s parametrem **`Ed.OFF`**.

Jste na řadě:

Napište program a stáhněte ho do Edisona. Budete muset pro Edisona vytvořit nějakou dráhu s tunely, kterými by mohl projíždět. Při jízde tunelem se zastíní vnější dopadající světlo a vy uvídíte, jak Edison rozsvítí svá přední světla.

Spusťte program a jeďte s Edisonem do tunelů, a uvidíte, jak to funguje.

Jméno _____

Potom zkuste experimentovat s číselnou hodnotou 100 v příkazu **if** na řádku 15, abyste zjistili, co se stane při spuštění programu s větším nebo menším číslem.

1. Co se stane, když hodnotu v příkazu **if** zvýšíte?

_____

2. Co se stane, když je hodnota v příkazu **if** nižší?

_____

Zkuste to!

Bude nějaký rozdíl, pokud místo funkce `ReadLeftLightLevel()` použijete funkci `ReadRightLightLevel()`? Upravte program s touto změnou a vyzkoušejte ho.

Jméno _____

Lekce 9: Pracovní list 9.3 – Sledování světla

V této činnosti napíšete program tak, aby Edison následoval světlo z obyčejné kapesní svítilny (baterky).

Podívejte se na tento program:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 while True:
14     if Ed.ReadRightLightLevel() - Ed.ReadLeftLightLevel() < 0:
15         Ed.Drive(Ed.FORWARD_LEFT, Ed.SPEED_5, Ed.DISTANCE_UNLIMITED)
16     else:
17         Ed.Drive(Ed.FORWARD_RIGHT, Ed.SPEED_5, Ed.DISTANCE_UNLIMITED)
18
```

Program porovnává úroveň světla mezi pravým a levým světelným čidlem a podle toho určuje, kudy program prochází.

Přítomnost světla baterky na levé nebo pravé straně robota způsobí, že robot přečte vyšší úroveň světla na příslušné straně. Logika tohoto programu říká, že když je úroveň světla zprava mínus úroveň světla zleva menší než nula, robot jede doleva směrem k silnějšímu zdroji světla, jinak robot pojede doprava.

Jste na řadě:

Úkol 1: Trasujte program

V pracovním listu 5.2 jste se dozvěděli o trasování programu. Pokud je v programu mnoho výpočtů a různých možných hodnot, může být užitečné program trasovat. To vám umožní pochopit různé hodnoty, které se mohou objevit, a můžete předvídat související chování.

1. Pro tento program vyplňte následující trasovací (sledovací) tabulku. Očekávané chování by mělo být buď 'jízda dopředu doprava' nebo 'jízda dopředu doleva'.

	Pravé světlo	Levé světlo	Očekávané chování
Baterka napravo	200	100	
Baterka nalevo	100	200	
Nesvítí	100	100	

Jméno _____

Úkol 2: Napište a spusťte program

Napište program a stáhněte ho do Edisona. Po stisknutí tlačítka přehrávání sviťte baterkou na Edisona. Robot pojede za světlem. Použijte baterku k ovládání směru Edisonovy jízdy.

2. Co se stane, když v programu změníte symbol 'menší než' (<) na symbol 'větší než' (>)?

_____

_____

Co myslíte?

Tento program sledování osvětlení demonstruje robotické chování, které se velmi podobá tomu, jak jsou můry v teplé noci přitahovány pouličním osvětlením.

Je můra přitahovaná světlem inteligentní? A co robot se stejným chováním?

Proč je hmyz, který přitahován světlem, naživu, ale robot není?

Jméno _____

Lekce 10: Pracovní list 10.1 – Robot upír

V této velmi náročné aktivitě použijete všechno, co jste se dosud naučili, a vytvoříte program k předvádění různého zajímavého chování Edisona, který se přemění na robota – upíra.

Abychom Edisona změnili na robota – upíra, použijeme objektově orientované programování.

Objekty a třídy v Pythonu

Python je objektově orientovaný programovací jazyk. Objektově orientovaný kód je silný způsob ke snižování složitosti uvnitř programu. To je důvod, proč mnoho programátorů používá objektově orientovaný kód ve velkých a komplikovaných programech.

V objektově orientovaném programovacím jazyce mohou programy kromě samostatných proměnných a funkcí používat také 'objekty'.

Objekt je množina funkcí a proměnných, které jsou ve vzájemném vztahu. Objekty jsou definovány 'třídou' (klíčové slovo Pythonu pro třídu je `class`).

Třída (nazývaná také ***definice třídy***) je jako plán, který popisuje, jak se má vytvořit objekt.

V programu můžete z třídy vytvářet spoustu různých objektů a pak těmito objekty ve programu jednotlivě manipulovat pro různé účely. Všechny objekty vytvořené z jedné třídy použijí ke svému vytvoření stejný "plán", ale pak můžete programem dělat s různými objekty různé věci.

Příklad: třída 'Shape'

Běžným příkladem třídy je **Shape** (česky Tvar).

Všechny tvary, např. čtverce, trojúhelníky a šestiúhelníky, mají společné atributy. Příklad podobnosti tvarů: Všechny tvary mají několik stran a určitý počet vrcholů. Existují také společné věci, které s tvary děláte, jako např. najít plochu nebo obvod tvaru.

Tyto společné prvky můžeme použít k vytvoření třídy pro **Shape** v Pythonu. Vytvořením třídy vytvoříme plán pouze se známými společnými prvky. Pak budeme schopni vytvářet z této třídy mnoho různých objektů, např. "čtverec" a "trojúhelník".

Syntaxe pro definici třídy v Pythonu je klíčové slovo `class` následované názvem, kterým chcete tuto třídu pojmenovat. Za tím je dvojtečka (:). Např.:

```
class Shape:
```

Všechno, co napíšete do odsazených řádků za slova `class Shape:`, se nachází uvnitř definice třídy.

Jméno _____

Podívejte se na následující příklad třídy pro Shape v Pythonu:

v Pythonu

```
class Shape:
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def area(self):
        return self.x * self.y

    def perimeter(self):
         return 2 * self.x + 2 * self.y
```

Tato třída říká, že plán pro každý objekt vytvořený z třídy Shape bude mít dva vstupní parametry (x a y). Třída také definuje tři funkce: 'init', 'area' a 'perimeter'.

Funkce __init__ a funkce self

Každá třída musí mít vždy funkci **__init__** .

Funkce __init__ je zahajovací (startovací) kód. Termín "init" znamená "inicializaci".

Když program potřebuje vytvořit objekt, program vchází do definice třídy a teprve tehdy spustí funkci __init__. Tato funkce obsahuje kód/kousek programu, který vytvoří objekt, což obvykle znamená nastavení počátečních hodnot proměnných objektu.

Funkce ve třídě musí mít jako svůj první parametr **'self.'** ('já sám'). Toto informuje nově vytvořený objekt, aby použil funkce a proměnné ***své vlastní***, patřící do tohoto objektu. (Tím je taky dáno, že se zabrání záměně případných stejných názvů proměnných, jak uvidíte níže v mých dalších poznámkách překladatele.)

Pozn. př. 1: Zápis se **self.** je podobný volání funkcí z knihovny **Ed**, to jsme zapisovali jako **Ed.** a připojili název funkce. ***Tečka*** za self nebo Ed se nesmí vynechat! A ještě jedna informace: **self** není klíčové slovo Pythonu.

Pozn. př. 2: Pro pochopení **self** z jiného pohledu, přepišme si výše uvedenou definici třídy takto (možná nám to trochu pomůže). Tento vysvětlující zápis ale dále obecně nepoužívejte! Nefungovalo by to s objektem, u kterého se objeví jeho nové jméno (třeba `rectangle` v dalším výkladu). Je to opravdu jen a jen výklad pro pochopení slova **self**: V nově vytvořeném objektu už ale **nebude toto slovo Shape** (jméno třídy, podle které se nový objekt zakládá), ale díky **self** se tam místo **self** (za sebe sama) bude automaticky dosazovat **název nového objektu**.

pro výklad

```
class Shape:
    def __init__(Shape,x,y):
        Shape.x = x
        Shape.y = y

    def area(Shape):
        return Shape.x * Shape.y

    def perimeter(Shape):
         return 2 * Shape.x + 2 * Shape.y
```

Pozn. př. 3: Uvědomte si prosím rozdíl mezi **x** a **Shape.x** – to první **x** je parametr předávaný z vnějšku do objektu třídy **Shape**, oproti tomu **Shape.x** je úplně něco jiného, je to tzv. atribut objektu – řečeno ne tak úplně přesně je to jakási vnitřní proměnná uložená v objektu. Viz tento jeden řádek ve třech pohledech na tu samou věc:

[1] (pro **výklad** třídy) `Shape.x = x`

[2] (přesně **v Pythonu**) `self.x = x`

[3] (**za běhu programu**; za běhu to již není "definice" třídy, ale nastává reálná práce nám skrytého kódu, který vytvoří konkrétní nový pojmenovaný **objekt**)

```
rectangle.x = 'vstupni_predany_parametr_1'
```

Řádek [1] v definici třídy, přesněji se musí použít ten [2], říká, že toto je **plán, předpis**, že až **budeme** pracovat s budoucím objektem, že **bude** mít vnitřní proměnnou x, to je levá strana od rovnítka (ano, je to nešikovná shoda jmen, vnitřní proměnná v **definici** třídy se pro lepší pochopení příkladu mohla jmenovat klidně třeba `stranaX`, ale toto 'neřešíme', protože překladač si je na rozdíl od programátora nikdy nepoplete), a tedy že vnitřní proměnná x převezme hodnotu **x** (to je to **x** vpravo do rovnítka), což je první parametr **později předaný reálnému objektu**, a říká, že je to to **x** popsané ve funkci `__init__(Shape,x,y)`, přesněji, v Pythonu se to ***musí*** psát `__init__(self,x,y)` .

Čili **self.x** je vnitřní proměnná nově založeného objektu -- a podle stejné třídy **Shape** těch objektů smíme založit v podstatě libovolně mnoho a každý z nich bude mít jiné paměťové místo pro svou proměnnou x (a taky pro y, pochopitelně)

Konec tří dlouhých poznámek překladatele --> podle mého názoru se však do jediného pracovního listu podařilo vtěsnat opravu hodně z myšlenek objektového programování. Python je, jak jinak, moderní objektový programovací jazyk, jehož zápis je 'lidsky' hezký, schůdný a jehož popularita díky tomu raketově vzrostla. K dalšímu studiu doporučuju materiály na internetu, jsou dostupné jak anglicky, tak česky – a já, 'jak jinak', doporučuju poprat se hned od začátku ***natvrdo*** s tou angličtinou!

Příklad: objekt obdélník (rectangle)

Když v našem programu vytvoříme definici třídy, můžeme ji hned použít k vytvoření objektů.

Řekněme, že chceme vytvořit objekt 'obdélník' ve třídě Shape.

Když chcete vytvořit nějaký objekt z třídy, v Pythonu, použijete syntaxi
object_name = class_name(parameters)
název_objektu = název_třídy(parametry)

Podívejte se na příklad vytvoření objektu 'obdélník' (již jen anglicky):

```
rectangle = Shape(100,45)
```

V programu toto (skrytě) zavolá funkci **__init__**, která skutečně proběhne a vytvoří objekt 'obdélník' (pojmenováváme ho **rectangle**) pomocí plánu z definice třídy **Shape**. Objekt 'obdélník' je sestaven tak, že hodnota **100** je vstupní hodnotou x a hodnota **45** je vstupní hodnotou y. (Nemusíte uvádět **self** - to je jen odkaz pro program, aby věděl, kde hledat.)

Jméno _____

Jakmile program vytvoří objekt, může přistupovat k funkcím tohoto objektu, které jsou definovány jeho třídou.

Chcete-li v Pythonu volat některou z těchto funkcí, použijete syntaxi
object_name.function_name(parameters)
název_objektu.název_funkce(parametry) *[nepřehlédněte tečku vevnitř!]*

Volání fuknce:

```
rectangle.area()
```

V tomto příkladu funkce **rectangle.area()** vrátí k již vytvořenému objektu obdélník (ten byl vytvořen s použitím vzoru/třídy/class `Shape` a má své **jednoznačné jméno** `rectangle`) hodnotu jeho vstupního parametru 'x' krát jeho vstupní parametr 'y'.

Pozn. př.: Hodnoty x a y jsou ale natvrdo zadané při vytvoření konkrétního obdélníka, neboli jednoho konkrétního objektu se jménem `rectangle` a je to 100 a 45. Hmmm... Tento příklad není příliš praktický. Kdybychm chtěli např. tři různé obdélníky, nemohly by mít stejné jméno. Na druhou stranu můžeme do jednoho `rectangle` **později dopravit jiné hodnoty** x a y a použít jeho vnitřní vzorce (zde se to zdá zbytečné, ale objekt může umět počítat i nějaké 'složitosti').

Jedním ze způsobů, jak tuto konkrétní funkci použít, je uložit její (`vrácenou` čili `return`) hodnotu do proměnné v nějakém programu. Např.:

AreaOfRectangle = rectangle.area ()

Nastavení třídy v EdPy

V EdPy v programu pro Edisona můžeme použít třídu k vytvoření různých objektů.

Zde je jednoduchá třída pro ***robota – upíra*** [třída **Vampire**] (s funkcí **__init__** a dvěma dalšími funkcemi: jednou funkcí pro denní chování robota a druhou pro noční chování. Třída **Vampire** také obsahuje proměnnou pro věk robota:

```
class Vampire:

    def __init__(self,age):
        self.age = age

    def doDayTimeBehaviour(self):
        Ed.LeftLed(Ed.OFF)
        Ed.RightLed(Ed.OFF)
        Ed.PlayTone(Ed.NOTE_A_7, self.age)
        while Ed.ReadMusicEnd() == Ed.MUSIC_NOT_FINISHED:
            pass
        Ed.TimeWait(1, Ed.TIME_SECONDS) # for debugging purposes and as a placeholder

    def doNightTimeBehaviour(self):
        Ed.RightLed(Ed.ON)
        Ed.LeftLed(Ed.ON) # for debugging purposes and as a placeholder
```

Uvedená definice třídy umožní vytvořit různé objekty Vampire a použít je v programu.

Jméno _____

Napište následující kód a ověřte si, zda rozumíte, co se v programu děje:

```
1
2  #-------------Setup----------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12
13 myEdisonVampire = Vampire(21)
14
15 while True:
16     # if it is the daytime
17     if Ed.ReadLeftLightLevel() > 100:
18         myEdisonVampire.doDayTimeBehaviour()
19     #it is night time
20     else:
21         myEdisonVampire.doNightTimeBehaviour()
22
23 class Vampire:
24
25     def __init__(self,age):
26         self.age = age
27
28     def doDayTimeBehaviour(self):
29         Ed.LeftLed(Ed.OFF)
30         Ed.RightLed(Ed.OFF)
31         Ed.PlayTone(Ed.NOTE_A_7, self.age)
32         while Ed.ReadMusicEnd() == Ed.MUSIC_NOT_FINISHED:
33             pass
34         Ed.TimeWait(1, Ed.TIME_SECONDS) # for debugging purposes and as a placeholder
35
36     def doNightTimeBehaviour(self):
37         Ed.RightLed(Ed.ON)
38         Ed.LeftLed(Ed.ON) # for debugging purposes and as a placeholder
39
```

Jste na řadě:

Úkol 1: Navrhněte chování upírů

Nyní nastala chvíle navrhnout ještě zajímavější chování pro robota – upíra. Buďte kreativní! Robot můžete reagovat na stisk tlačítka, tlesknutí a překážky. Uvažujte o tom, že byste v programu použili události. Můžete vytvořit vícenásobné objekty upíra s různou volbou věku. Můžete také upravit základní program tak, aby vaše objekty měly více vstupních parametrů.

Promýšlejte, co všechno chcete do svého programu zahrnout, a pak pracujte na návrhu programu tak, že nejprve vytvoříte vývojové diagramy a pak pseudokód na základě vývojových diagramů.

Jméno _____

Krok 1: Vývojové diagramy

Nakreslete vývojové diagramy pro denní a noční chování buď ručně, nebo pomocí programu jako je Google Slides nebo podobný kreslící program. Vývojové diagramy nakreslete sem. Pokud potřebujete, použijte další papír.

Jméno _____

Krok 2: Pseudokód

Ručně 'přeložte' vývojové diagramy do pseudokódu. Svůj pseudokód napište sem. Pokud potřebujete, použijte další papír.

Jméno _____

Úkol 2: Napište svůj program

Přeložte svůj pseudokód do kódu Pythonu pomocí aplikace EdPy. Stáhněte si ho do svého robota Edison a spusťte svůj program a vyzkoušejte ho.

Úkol 3: Řešení problémů

Při programování je zcela běžné, že něco nefunguje tak, jak si myslíte, že to bude fungovat. Žádný problém! Pamatujte si na toto rčení: Thomas Edison slavně selhal 10 000krát, než úspěšně vynalezl žárovku. Takže, snažte se dál!

Učíme se z problémů, s nimiž se setkáváme, takže je dobrým zvykem věnovat čas přemýšlení o tom, co se stalo. Popište všechny problémy, které jste měli, na jaké nápady jste přišli při jejich opravách a popište řešení, která jste nalezli v pozdějších pokusech. Pokud potřebujete, použijte další papír.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Jméno _____

Úkol 4: Popište některé programové struktury ve svém programu

Vyberte tři různé programové struktury, které jste použili ve svém programu. Popište, co každá dělá:

1. Název programové struktury:

_____

Co dělá?

_____

_____

_____

_____

2. Název programové struktury:

_____

Co dělá?

_____

_____

_____

_____

3. Název programové struktury:

_____

Co dělá?

_____

_____

_____

Úkol 5: Představte svůj program

Ukažte svého robota upíra svému partnerovi, skupině nebo třídě. Mluvte o svých nápadech, programu, problémech, se kterými jste se setkali, a jak jste je vyřešili.

[-~~ konec ~~-]