Obsah

Kapitola 1

Robot, robotika

1.1 Robot

Slovo *robot* po prvýkrát použil český spisovateľ Karel Čapek vo svojej hre R.U.R (Rossum's Universal Robots). Je odvodené od slova "robota" aj keď niektorí tvrdia, že je odvodené od jeho slovenského významu a teda *robot* vlastne pochádza zo Slovenska.

1.1.1 Základné časti robota

Robot sa väčšinou skladá z :

motorickej časti má na starosti pohyb robota

senzorickej časti tvoria ju senzory, vďaka ktorým je schopný získavať informácie z prostredia a na základe nich sa rozhodovať

riadiacej časti zbiera údaje zo senzorickej časti, na základe nich sa rozhoduje a ovláda motorickú časť

1.2 Zákony robotiky

Zákony robotiky definoval v roku 1942 v poviedke "Runaround" Isaac Asimov. Znejú takto:

1. **Robot nesmie ublížiť človeku alebo svojou nečinnosťou dopustiť, aby mu bolo ublížené.**
2. **Robot musí poslúchnuť človeka, okrem prípadov, keď je to v rozpore s prvým zákonom.**
3. **Robot sa musí chrániť pred poškodením, okrem prípadov, keď je to v rozpore s prvým alebo druhým zákonom.**

Tieto zákony veľmi dobre určujú ako by sa mali roboty správať, ale mohlo by sa stať, že nebudú správne pochopené [1] . Preto pán Asimov pridal ešte takzvaný "nultý" zákon, ktorého znenie je:

[1] To sa stalo námetom pre sci-fi žáner. Príkladom môže byť film *Ja, Robot*, kde roboti v súlade s prvým zákonom nechceli, aby si ľudia navzájom ubližovali a preto chceli nastoliť najlepšiu formu vlády – diktatúru. Tým síce vyhoveli všetkým trom zákonom (ba dokonca i nultému), ale v konečnom dôsledku ľuďom uškodili.

0. Robot nesmie ublížiť ľudstvu, alebo svojou nečinnosťou dopustiť aby mu bolo ublížené.

a je nadradený ostatným trom zákonom.

Smutným faktom je, že absolútna väčšina robotov s ktorými sa stretneme sa týmito zákonmi nebude riadiť. Dôvody prečo necháme na čitateľa.

1.3 Krátky úvod do histórie robotiky

1.4 Robotika

Kapitola 2

Lego Mindstorms NXT a nxtIDE

Kapitola 3

Prvý program

Hádam v každej knihe, ktorá predstavuje nový programovací jakyk je prvá kapitola venovaná zrejme najtriviálnejšej veci ktorá v programovaní existuje a tou je vypísanie vety "Hello World!"[1] na nejaké výstupné zariadenie. Túto tradíciu porušíme, pretože vypisovanie textu na obrazovku NXT kocky je celkom náročné a nieje vhodné hneď do prvej kapitoly.

Namiesto toho hneď začneme písaním programu. Otvorte si `nxted` a prepíšte doň nasledujúci text:

```
def main():                 ❶
    OnFwd(OUT_A, 60)        ❷
    OnFwd(OUT_B, 60)        ❸
    Wait(2000)              ❹
    OnRev(OUT_AB, 60)       ❺
    Wait(2000)              ❻
    Off(OUT_AB)             ❼
```

***Upozornenie** : čísla na konci riadkov neprepisujte, niesú súčasťou programu. Slúžia iba na jednoduchšiu orientáciu pri jeho vysvetľovaní.*

Ak ste program prepísali, uložte ho po názvom "pr1.py" (**File** -> **Save**) . Potom ho spustite v emulátore kliknutím na **Run** -> **Run in nxtemu** (alebo použite klávesovú skratku **F6**).

Ak sa robot v emulátore chvíľu pohyboval vpred, potom vzad a nakoniec zastal na rovnakom mieste z akého začal, potom je na mieste zagratulovať. Práve sa vám podarilo napísať a spustiť prvý program pre robota v Pythone.

Je veľmi pravdepodobné, že sa vám program pri prepisovaní zdal komplikovaný a vôbec ste mu nerozumeli. Žiadne strachy, rozoberieme ho riadok po riadku.

❶ Programy pre NXT kocku sa skladajú z funkcií. V našom programe máme iba jednu funkciu a to funkciu `main`. Táto funkcia je špeciálna, pretože sa spustí hneď po zapnutí programu a je tiež jediná, ktorú musí obsahovať každý program, ktorý chce bežať v NXT kocke.

[1] po slovensky "Ahoj Svet!"

Samotné funkcie sa skladajú z príkazov. V našom programe máme 6 príkazov, každý na samostatnom riadku. Každý z nich je odsadený. To z nich robí jeden celok – blok kódu, ktorý tvorí telo funkcie. Konkrétne v našom prípade používame na odsadenie 4 medzery. Nemusia to byť vždy 4 medzery, ale odsadenie musí byť v celom programe rovnaké.

NEPODLIEHAJTE PANIKE ! – Ak ste posledný odstavec vôbec nepochopili, vôbec sa tým netrápte. Zatiaľ stačí, aby ste vedeli ako zadefinovať funkciu `main` a že každý príkaz musí byť odsadený. Všetko ostatné podrobne preberieme v ďalších častiach.

❷ Príkaz OnFwd hovorí robotovi, aby zapol motor. Konkrétne na tomto riadku hovorí, že má zapnúť motor, ktorý je pripojený na výstup A. Robot sa to dozvie podľa konštanty `OUT_A`. Za čiarkou nasleduje číslo 60. To hovorí robotovi, aby motor na výstupe A zapol na 60% výkonu.

❸ Na tomto riadku je znova použitý príkaz OnFwd. Tentokrát však zapína monot na výstupe B. Rovnako ako na predchádzajúcom riadku i tu nastavuje rýchlosť na 60%.

❹ Vo chvíli, keď sa robot dostane vo vykonávaní dostane až sem, má už zapnuté motory na výstupoch A a B na 60%. Keďže chceme aby sa pohyboval vpred, necháme kocku chvíľu oddychovať – prikážeme jej chvíľu počkať.

Zabezpečíme to príkazom `Wait`. Príkaz `Wait` hovorí robotovi aby istý čas počkal s vykonávaní príkazov. Tento čas sa zadáva v tisícinách sekundy, teda milisekundách.

V našom prípade hovoríme robotovi aby počkal 2000 milisekúnd, teda 2 sekundy.

❺ Robot sa zatiaľ pohyboval 2 sekundy dopredu. Teraz chceme, aby sa pohyboval rovnaký čas vzad. Presne na toto slúži príkaz `OnRev`. Robí presne to isté, čo príkaz OnFwd iba s tým rozdielom, že nastavuje motorom opačný smer.

Príkaz na tomto riadku teda hovorí robotovi, aby zapol motory na výstupoch A a B na 60% smerom vzad.

❻ Keďže chceme, aby sa robot pohyboval vzad rovnako dlho ako vpred, necháme ho ísť tiež 2 sekundy s použitím funkcie `Wait`.

❼ Aj keď by sme si mohli myslieť, že tento riadok je už nieje potrebný, opak je pravdou. Keby sme program ukončili už na predchádzajúcom riadku, motory by po jeho skončení už síce nedostávali žiadnu energiu, ale robot by sa zotrvačnosťou stále pohyboval vpred (i keď stále menej a menej).

Preto je potrebné motory zabrzdiť. To aj na tomto riadku robíme, konkrétne brzdíme motory A a B pomocou funkcie `Off`.

Názov výstupu	**Význam**
OUT_A	Výstup A
OUT_B	Výstup B
OUT_C	Výstup C
OUT_AB	Výstupy A a B
OUT_BC	Výstupy B a C
OUT_ABC	Výstupy A, B a C

Cvičenie č. 1 Na riadkoch ❷ a ❸ sú použité dva príkazy `OnFwd`. Na riadku ❺ je použitý príkaz `OnRev`, ktorý má presne opačný efekt, avšak zaberá iba jeden riadok. Spojte riadky ❷ a ❸ do jedného s použitím konštanty OUT_AB.

Cvičenie č. 2 Experimentujte s programom zo zadania. Skúste vynechať niektoré časti programu, sledujte čo sa stane a skúste prísť nato prečo.

Cvičenie č. 3 Upravte program zo zadania tak, aby sa robot chvíľu pohyboval vpred, potom zostal chvíľu stáť a nakoniec sa vrátil na miesto odkiaľ štartoval.

Kapitola 4

Pohyb robota II

Pohyb robota vpred a vzad je veľmi zaujímavá vec. Naposledy sme motory púštali na 60% výkonu. Čo tak vyskúšať akou maximálnou rýchlosťou sa robot vie pohybovať? Žiaden problém, nastavíme výstupy na 100% výkonu. Ak by sme chceli upraviť predchádzajúci program, museli by sme všade hodnotu 60 nahradiť hodnotou 100. Neustále prepisovanie rovnakých častí programu je ale strašne nudné. A keď sa programovanie stane nudným, je to jasným signálom, že je niečo zle a že existuje lepší spôsob.

4.1 Konštanty

Ak sa pozrieme na predchádzajúci program pozornejšie, môžeme si všimnúť, že číslo 60 v ňom má rovnaký význam. Určuje na aký výkon budú výstupy A a B nastavené. A ak sa pozrieme ešte pozornejšie, tak zistíme, že sa v programe opakuje aj hodnota 2000. Tá určuje, ako dlho pôjde robot vpred a potom vzad.

Môžeme teda tieto hodnoty nazvať menom, ako napríklad v programe nižšie.

```
VYKON = 100                 ❶
CAS_POHYBU = 2000           ❷

def main():
    OnFwd(OUT_AB, VYKON)    ❸
    Wait(CAS_POHYBU)        ❹

    OnRev(OUT_AB, VYKON)    ❺
    Wait(CAS_POHYBU)        ❻

    Off(OUT_AB)
```

❶ Ak nejakú hodnotu nazveme menom, vytvoríme (tak ako vo Fyzike) konštantu. Na tomto riadku vytvárame konštantu `VYKON`. Označuje na akú veľkosť budú výstupy nastavené.

❷ Na tomto riadku tiež vytvárame konštanu, konkrétne `CAS_POHYBU`, ktorá označuje ako dlho sa bude robot pohybovať vpred a vzad.

Vo všeobecnosti sa konštanty skladajú z veľkých písmen anglikej abecedy, čísel a podtrhovníka (_), pričom nezačínajú číslom. Podtrhovník sa používa na oddeľovanie slov v dlhších názvoch, ako napríklad `CAS_POHYBU`.

❸, ❹, ❺ a ❻ Potom ako je konštanta definovaná, zmení sa jej názov všade za hodnotu, na ktorú bola nastavená. Preto má v tomto prípade riadok ❸ rovnaký význam ako `OnFwd(OUT_AB, 100)`. Podobne to funguje aj na riadkoch ❹, ❺ a ❻.

Cvičenie č. 1 Experimentujte z programom zo zadania. Skúste nastaviť konštanty na rôzne (aj záporné) hodnoty a pozorujte, čo sa zmení.

Cvičenie č. 2 Prepíšte všetky predchádzajúce cvičenia tak, že namiesto opakujúcich sa hodnôt použijete konštanty.

4.2 Zatáčanie a komentáre

Aj keď je pohyb robota vpred a vzad je veľmi zaujímavá vec, je oveľa zábavnejšie, keď sa robot pohybuje aj do rôznych iných smerov.

V tejto časti si ukážeme, ako robota otočiť o 90 stupňov. Máme dve možnosti:

1. Necháme točiť iba jeden motor a druhý vypneme.
2. Jedným motorom budeme točit do jednej strany a druhým do opačnej. Tým sa budeme točiť prakticky na mieste a celý pohyb bude trvať kratšie.

Druhá možnosť sa nazýva aj divergentný pohyb a je tiež o niečo náročnejšia.

Ukážeme si preto prvú možnosť. Robot pôjde najprv chvíľu vpred a potom sa pokúsi otočiť o 90 stupňov.

```
VYKON = 70
CAS_VPRED = 900
CAS_90_STUPNOV = 2260      ❶

def main():
    # pohyb vpred          ❷
    OnFwd(OUT_AB, VYKON)
    Wait(CAS_VPRED)
    Off(OUT_AB)

    # otocka o 90 stupnov  ❸
    OnFwd(OUT_A, VYKON)
    Wait(CAS_90_STUPNOV)
    Off(OUT_A)
```

❶ Konštanta `CAS_90_STUPNOV` udáva ako dlho sa má jeden z motorov pri výstupe zapnutom na 70% (konštanta `VYKON`) točiť, aby otočil celého robota o 90 stupňov. Pre `nxtemu` má hodnotu 2260. Je ale možné, že túto hodnotu budete musieť upraviť v závislosti od vášho počítača či robota.

❷ **a** ❸ Všetko, čo sa v Pythone nachádza za mriežkou (#)[1] sa nazýva komentár. Komentáre sa používajú na popísanie jednotlivých častí programu, ktoré niesú očividné. Môžu byť napísané v akomkoľvek jazyku od čínštiny cez fínštinu až po slovenčinu, pretože sú pri vykonávaní programu ignorované.

Tento program sa od všetkých predchádzajúcich odlišuje. Je o čosi dlhší a ak by sme rátali prázdne riadky a medzery, určite by sme ich narátali viac. Tiež sme sa v ňom prvý krát stretli s komentármi.

Ak sa na druhej strane pozreme na úplne úplne prvý program (o kapitolu dozadu) a porovnáme ho s týmto, môže sa stať, že sa nám ten prvý bude zdať úplne škaredý. A to je dobre. Pri programovaní si treba vždy pamätať, že kód príšeme raz, ale čítať ho budeme (či už my alebo niekto iný) mnohokrát. Programovanie má byť zábava, ale ak sa dostaneme ku kódu, ktorému vôbec nerozumeieme (a ešte ten kód nedajbože nieje cudzí) je po zábave a to je signálom, že je niekde problém a treba ho riešiť.

Ten problém je naštastie riešitelný a to dokonca veľmi jednoducho. **Píšte komentáre.** Budúce generácie (a neskôr aj vy sami) vám budú vďačné.

Cvičenie č. 3 Upravte všetky vaše programy tak, aby boli čitateľné. Na miesta, ktoré niesú okamžite zrozumiteľné pridajte komentáre. Časti programov ktoré spolu súvisia oddeľte prázdnymi riadkami.

Cvičenie č. 4 Prepíšte program zo zadania tak, aby robot namiesto zatáčania jedným kolesom použil divergentný pohyb oboch kolies.

Pomôcka: Ak pôjdu oba motory rovnako rýchlo, musí im stačiť polovičný čas.

[1] Myslí sa tým od mirežky vpravo

Kapitola 5

Opakovanie príkazov

5.1 Cyklus for

Existuje viacero možností ako zopakovať niekoľko príkazov viac krát. Mohli by sme napríklad príkazy skopírovať a potom vložiť toľko krát, koľko by bolo treba. To však nieje ani pekné, ani rozumné. Prvou nevýhodou je, že je v takom kóde zmätok, preože nieje zrejmé, koľko krát sa bude vykonávať. Ak by sme chceli niekedy neskôr pridať ešte viac opakovaní, museli by sme kód znova kopírovať, čo začne byť po chvíli nuda. A keď je pri programovaní nuda, potom niečo nieje v poriadku.

V Pythone naštastie existuje aj iná možnosť a to cyklus for. Jeho použitie si vysvetlíme na príklade pohybu robota do štvorca.

```
VYKON = 70
CAS_VPRED = 900
CAS_90_STUPNOV = 2260

def main():
    # zopakujeme 4 krat
    for x in range(0, 4):          ❶
        # pohyb vpred
        OnFwd(OUT_AB, VYKON)
        Wait(CAS_VPRED)
        Off(OUT_AB)

        # otocenie o 90 stupnov
        OnFwd(OUT_A, VYKON)
        Wait(CAS_90_STUPNOV)
        Off(OUT_A)
```

❶ Celý tento riadok by sme mohli preložiť do slovenčiny ako

```
pre každé x v rozpätí od 0 do 4 vykonaj nasledujúce príkazy
```

Možno vás zarazilo, že v rozpätí od 0 do 4 sú predsa čísla 0, 1, 2, 3 a 4 – dohromady päť čísel. Je tomu naozaj tak, avšak funkcia `range` (mimochodom `range` je funkcia rovnako ako napríklad `Wait`) chápe rozpätie trochu inak. Prvý argument tejto funkcie označuje od akého čísla treba začať počítať (v našom prípade od 0) a druhý označuje dokedy treba počítať. Posledné číslo, po ktoré dopočíta musí byť menšie ako druhý argument. Preto v našom prípade vygeneruje iba čísla 0, 1, 2 a 3.

Aby sa programovanie zjednodušilo, pozná Python aj skrátený tvar pre zápis tohto riadku. V prípade že chceme začať počítať od 0, stačí použiť iba jeden argument a to dokedy treba počítať. Tento riadok by sme teda mohli prepísať ako

```
for x in range(4):
```

5.2 Špirála

Jednou z dôležitých vlastností cyklu for je to, že pokiaľ prechádza[1] nejaké rozmedzie hodnôt, aktuálnu hodnotu si pamätá.

To sa dá veľmi dobre využiť. Ukážeme si program, v ktorom sa robot bude pohybovať v špirále.

```
VYKON = 70
CAS_VPRED = 900
CAS_90_STUPNOV = 2260

def main():
    # zopakujeme 10 krat
    for x in range(10):                ❶
        # pohyb vpred
        OnFwd(OUT_AB, VYKON)
        Wait(x * CAS_VPRED)            ❷
        Off(OUT_AB)

        # otocenie o 90 stupnov
        OnFwd(OUT_A, VYKON)
        Wait(CAS_90_STUPNOV)
        Off(OUT_A)
```

❶ Budeme prechádzať čísla od 0 po 9 – dokopy 10 čísel.

❷ Na tomto riadku sa oproti predchádzajúcemu programu zmenil výraz v zátvorke na `x * CAS_VPRED`.

Počas jednotlivých opakovaní príkazov nadobúda `x` zakaždým o 1 väčšiu hodnotu. Najprv bude mať hodnotu 0, potom sa zvýši na 1, potom na 2 a takto až po 9.

Hodnotou, ktorú má aktuálne `x` bude potom vynásobená konštanta `CAS_VPRED`, čo spôsobí, že sa robot bude pohybovať vpred stále dlhšie a dlhšie. V konečnom dôsledku sa bude pohybovať po špirále.

[1] po programátorsky sa povie "iteruje cez"

Kapitola 6

Pohyb robota III

Pohybovať sa vpred a vzad či dokonca zatáčať dokáže akékoľvek auto. Avšak rozdiel medzi motorom, ktorý poháňa auto a motormi, ktorými je poháňaný robot je v presnosti. Motory, ktoré má robot dokážu dostať z motorov naspäť informáciu o tom, koľko otáčok naozaj motor spravil. Vďaka tomu vieme potom v programe zaručiť, že sa koleso otočí vždy rovnako ďaleko. Slúži na to funkcia RotateMotor.

6.1 Presný pohyb s `RotateMotor`

Funkcia `RotateMotor` prijíma tri argumenty. Prvý je výstup, na ktorý je motor pripojený. Nasleduje rýchlosť na akú chceme motor nastaviť. Posledným je počet otáčok o ktoré chceme motor otočiť. Ako funkciu použiť si ukážeme na nasledujúcom príklade.

```
OTACKY_VPRED = 100
OTACKY_90_STUPNOV = 180
VYKON = 60

def main():
    #  pohyb dopredu
    RotateMotor(OUT_AB, VYKON, OTACKY_VPRED)      ❶

    # otocenie o 90 stupnov
    RotateMotor(OUT_A, VYKON, OTACKY_90_STUPNOV)  ❷
```

Zaujímavé na tejto funkcii je aj to, že výkon i otáčky môžu mať aj zápornú hodnotu. V takom prípade sa budú motory pohybovať dozadu.

Cvičenie č. 1 Napíšte program, vďaka ktorému sa bude robot pohybovať do štvorca.

6.2 Náhodný pohyb

Pohyb vpred a vzad je naozaj zaujímajvá vec. Je ale omnoho zaujímavejšie sledovať robota ako robí čo sme od neho nechceli. Slúži nám na to funkcia `Random`, ktorá vráti náhodné číslo. Takéto

náhodné číslo potom môžeme posunúť napríklad funkcii `Wait` či `RotateMotor` a nechať tak robota pohybovať sa náhodne.

Môžeme si to ukázať na nasledujúcom programe, vďaka ktorému sa bude robot pohybovať po takzvanej "náhodnej špirále". Namiesto toho, aby sa pohyboval zakaždým viac a viac vpred, pohne sa vpred o náhodnú vzdialenosť.

```
OTACKY_VPRED = 50
OTACKY_90_STUPNOV = 180
VYKON = 60

def main():
    for x in range(10):
        #  pohyb dopredu
        RotateMotor(OUT_AB, VYKON, OTACKY_VPRED * Random(6))  ❶

        # otocenie o 90 stupnov
        RotateMotor(OUT_A, VYKON, OTACKY_90_STUPNOV)
```

❶ Tento riadok zabezpečuje aby sa robot pohol vpred o náhodnú vzdialenosť. Dôležitá je časť `Random(6)`. Tá vygeneruje pri každom volaní nové náhodné číslo[1] rozsahu 6, čo v preklade znamená od 0 do 5 (0, 1, 2, 3, 4, 5 – teda 6 čísel)[2]. Aby bolo vidieť rozdiel, je toto číslo potom vynásobené konštantou `OTACKY_VPRED`.

Môže sa však stať, že sa náhodne vygeneruje nula. Potom sa robot nepohne vpred vôbec a urobí otáčku o 180 stupňov.

Cvičenie č. 1 Experimentujte s hodnotou konštanty `OTACKY_VPRED`. Čo sa stane ak bude záporná?

[1] To že je nové neznamená, že sa nemôže stať, že vygeneruje za sebou dve rovnaké

[2] Ak sa vám to zdá extrémne podobné s príkazom **for**, ste na dobrej ceste.