IB113 Úvod do programování a algoritmizace

Cvičení 5

Jaromír Plhák

Domácí úkol 1

- Hodnocení v poznámkových blocích
- Čtěte pozorně zadání a nemažte jej ani testy
  - Pep8
  - Neměnit hlavičky funkcí
- print vs. return
- odd_divisor
  - Často vynechán dělitel roven n (chyba v rozsahu řídící proměnné for cyklu)
- tribonacci
  - Často řešeno zbytečně složitě (if n == 0, if n == 1, if n == 2)

Domácí úkol 1

- number_of_digits
  - Více možností, většinou bez problému
- revert_string
  - Více možností, většinou bez problému
- from_dec_to_bin
  - Špatný výstup pro nulu
  - Pro obrácení řetězce šla výhodně využít funkce revert_string

Zadání domácí úlohy 2 – Člověče nezlob se

- V ISu v odevzdávárně
  - https://is.muni.cz/auth/el/1433/podzim2017/IB113/ode/72006396/72006443/hw02_zadani.py (skupina 1)
  - https://is.muni.cz/auth/el/1433/podzim2017/IB113/ode/72006400/72006457/hw02_zadani.py (skupina 2)
- Veškeré informace v souboru
- Odevzdejte pouze tento soubor do odevzdávárny
  - https://is.muni.cz/auth/el/1433/podzim2017/IB113/ode/72006396/72006430/ (skupina 1)
  - https://is.muni.cz/auth/el/1433/podzim2017/IB113/ode/72006400/72006451/ (skupina 2)
- Do **čtvrtka** 26. 10. 2017, 23:59

Úkoly

- 4.2.1. Opilec na cestě domů
  - Opilec je na půli cesty mezi domovem a hospodou, každý krok udělá náhodně jedním směrem. Napište funkci, která bude simulovat opilcův pohyb. Parametry budou vzdálenost mezi domovem a hospodou a počet kroků do opilcova usnutí (tj. maximální délka simulace). Simulace skončí buď tehdy, když opilec dojede domů nebo do hospody, případně po vyčerpání počtu kroků.
- 4.2.2. Analýza opilce
  - V tomto příkladu tedy použijeme předchozí program pro jednoduchou analýzu, jak to dopadne, když to zkusíme zopakovat vícekrát za sebou.
  - Nejprve upravte funkci z předchozí příkladu tak, aby nevypisovala stav opilce (například přidáním volitelného parametru output a zapodmínkováním výpisu) a aby vracela True dojde-li opilec domů a False pokud ne.
  - Následně napište funkci, která provede simulaci opilce count-krát a vypíše procentuální úspěšnost dojití domů.

Řetězec (string)

- Sekvence znaků (čísla, písmena, symboly) uzavřených:
  - a) V apostrofech: my_string = 'Brian'
  - b) V úvozovkách: my_string = "Brian"
- Pokud má řetězec obsahovat apostrof?

```
>>> print('I'm Brian !')
SyntaxError : invalid syntax
>>> print('I\'m Brian !') # Escaping
I'm Brian !
>>> print("I'm Brian !")
I'm Brian !
```

Délka řetězce

- Funkce len():

```
>>> example = "Python"
>>> len(example)
6
```

- Prázdný řetězec = prázdné apostrofy / uvozovky:

```
>>> empty = ""
>>> type(empty)
<class 'str '> # It really is a string
>>> len(empty)
0
```

Jednotlivé znaky řetězce

- Jednotlivé znaky řetězce jsou indexované celými čísli
  - Indexuje se od 0 po len(string) - 1!

>>> example = "Python"

Znak	P	y	t	h	o	n
Index	0	1	2	3	4	5

- Syntaxe zpřístupnění znaků: string[index]

>>> character = example[1]

>>> character

'y'

- Řetězce jsou neměnitelné (*immutable*): nefunguje např. example[1] = 'i'!

Rozsah indexů (řezy)

Znak	P	y	t	h	o	n
Index	0	1	2	3	4	5

```
>>> example [1:4] # starting + ending index
'yth'
>>> example [1:] # starting only , default ending index is the length of string
'ython'
>>> example [:4] # ending only , default starting index is 0
'Pyth'
```

Rozsah indexů (řezy)

Znak	P	y	t	h	o	n
Index	0	1	2	3	4	5

```
>>> example [ -4: -1] # starting and ending
'tho'
>>> example [ -4:] # starting only
'thon'
>>> example [: -1] # ending only
'Pytho‘
>>> example [::2] # string[start:end:step]
'Pto'
```

Iterace nad řetězcem

- Cyklus for

```
string = "spam!"
for character in string:
    print (character)
```

- Proměnná character postupně nabývá hodnoty všech znaků řetězce
- Druhá možnost:

```
string = "spam!"
for i in range(len(string)):
    print(string[i])
```

Aritmetické operátory nad řetězcem

- +

```
>>> "King " + "Arthur " + str (1) + "st"
'King Arthur 1st'
```

- *

```
>>> "Five" * 3
'FiveFiveFive'
```

Operátory in, not in

- Test na výskyt znaku / podřetězce v retězci

```
>>> 'p' in "Python"
False
>>> 'r' not in "Python"
True
>>> 'tho' in "Python"
True
>>> 'toh' in "Python"
False
```

Funkce nad řetězci

- Syntax: string.function()
- Nemění původní retězec
  - lower() - vrátí řetězec s malými písmeny místo velkých
  - upper() - vrátí řetězec s velkými písmeny místo malých

```
>>> reaction = "What The Hell?"
>>> reaction.lower()
'what the hell?'
>>> reaction.upper()
'WHAT THE HELL?'
>>> reaction
'What The Hell?'
```

Funkce nad řetězci

- count(s) - spočítá výskyt řetězce s v daném retězci

```
>>> yoda = "Do or do not... There is no try."
>>> yoda.count("o")
5
>>> yoda.count("no")
2
>>> yoda.count("do")
1
```

Funkce nad řetězci

- find(s) - najde první index výskytu řetězce *s* v daném řetězci

```
>>> yoda = "Do or do not... There is no try."
>>> yoda.find("o")
1
>>> yoda.find("no")
9
>>> yoda.find("do")
6
>>> yoda.find("x")
-1
```

Funkce nad řetězci

- replace(a, b) - vrátí nový řetězec, který v daném řetězci nahradí všechny výskyty řetězce a řetězcem b

```
>>> yoda = "Do or do not... There is no try."
>>> yoda.replace("o", "0")
'D0 0r d0 n0t... There is n0 try.'
>>> yoda . replace("no", "nooo")
'Do or do nooot... There is nooo try.'
>>> yoda.replace("There is no try.", "No try, there is.")
'Do or do not... No try, there is.'
>>> yoda.replace("anakin", "vader")
'Do or do not... There is no try.'
```

Funkce nad řetězci

- split() - rozdělí řetězec podle mezer nebo specifikovaného oddělovače
  - Výsledkem je seznam řetězců
  - Oddělovač se v průběhu procesu zahazuje

```
>>> yoda = "Do or do not... There is no try."
>>> yoda.split()
['Do ', 'or ', 'do', 'not... ', 'There', 'is', 'no', 'try.']
>>> yoda.split('o')
['D', ' ', 'r d', ' n', 't... There is n', ' try.']
```

Funkce nad znaky

- Znak je řetězec délky 1
- **ASCII**: http://www.ascii-code.com/
- ord(c), kde *c* je znak
  - Vrací celočíselný ASCII kód znaku
- chr(i), kde *i* je celé číslo
  - Vrací znak, jehož ASCII kód je *i*

```
>>> ord("A")
65
>>> chr(65)
'A'
```

Úkoly

- 5.2.2. Zdvojení písmen
  - Napište funkci, která vrátí nový řetězec, ve kterém bylo každé písmenko zdvojeno.
- 5.2.6. Znaky na stejných pozicích
  - Napište funkci, která dostane dva řetězce a vypíše ty znaky, které jsou na shodných pozicích stejné.
- 5.2.8. Palindrom
  - Napište funkci, která vrátí, zda je řetězec palindromem. Palindromem je takové slovo či věta, která má při čtení v libovolném směru stejný význam, například nepotopen či jelenovi pivo nelej (mezery můžete ignorovat).
- 5.2.9. Hodnota slova
  - Každý znak A-Z má hodnotu 1-26 (diakritiku a velikost písmen pro tento příklad ignorujte). Napište funkci, která spočítá a vrátí hodnotu vloženého řetězce (slova).