Python

12.11.2021

Mária Šoltésová
maria.soltesova@mff.cuni.cz
(maria.soltesova@gmail.com)

Programovací jazyk

- **1. generace jazyků - strojový kód**
  - Procesor počítače: instrukce (sekvence bitů) v hexadecimální soustavě
  - Instrukce umožňují pouze např. sčítání adres nebo skoky mezi instrukcemi
  - Instrukce se procesoru předloží v binární podobě
  - Strojový kód je tedy extrémně nečitelný a závisí na instrukční sadě daného CPU
  - Každý program musí být do strojového kódu přeložen
- **2. generace jazyků - assembler**
  - Instrukce v něm mají slovní označení (kód)
  - Kódy instrukcí se poté přeloží na výše uvedený strojový kód
- **3. generace jazyků**
  - Zaměřují se na to, by byl program čitelný z hlediska člověka
  - Čísla jsou vnímána již jako proměnné, zdrojový kód připomíná matematický zápis

sčítání dvou čísel:

strojový kód:

```
2104
1105
3106
7001
0053
FFFE
0000
```

Assembler:

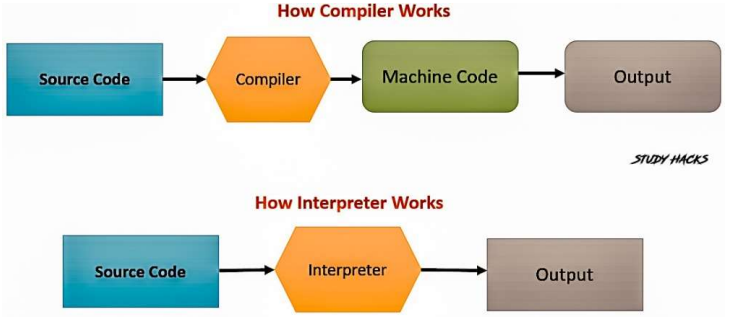
```
ORG
100
LDA A
ADD B
STA C
HLT
DEC 83
DEC -2
DEC 0
END
```

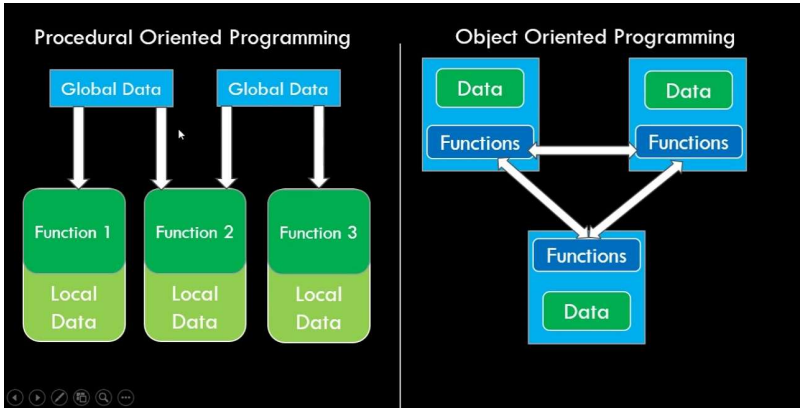
C:

```
int main(void)
{
    int a, b, c;
    a = 83;
    b = -2;
    c = a + b;
return 0;
}
```

Programovací jazyk

- **Jazyky**
  - **Kompilované jazyky:** zdrojový kód překládá do strojového kódu pomocí překladače (kompiler) celý najednou
  - **Přímo interpretované jazyky:** kód se překládá až za běhu programu
- **Programování**
  - **Objektovo orientované (OOP):** umožňuje programovat aplikace velmi přehledně tak, že je rozdělíme do objektů, které spolu komunikují. Naprostá většina moderních jazyků OOP využívá.
  - **Funkcionálně orientované:** program je rozdělen na menší části (podproblémy), které se řeší samostatně pomocí funkcí





Úvod do Pythonu

- Programovací jazyk, autor: Guido van Rossum, 1991
- Open-source
- Použití:
  - Vývoj softwaru
  - Vývoj webů
  - Matematika
  - Systémové skripty
- Výhody:
  - Pracuje na různých platformách
  - Dynamicky interpretovaný jazyk
  - Jednoduchá a čistá syntaxe
  - Podporuje procedurální, funkcionální a objektové paradigma
  - Jádro a náročné funkce napsané v C
- Nejnovější verze: Python 3.10.0, www.python.org
- IDE (integrated development environment)

Základní operace v Pythonu

Operátor	**Zápis**	**Popis**	**Příklad**
Sčítání	+		1 + 2 = 3
Odčítání	-		3 - 1 = 2
Násobení	*		2*3 = 6
Dělení	/	Vrátí desetinné číslo	12/4 = 3.0
Modulo	%	Vrátí zbytek po dělení	5%2 = 1
Mocnina	**	x**y	2**3 = 8
Celočíselné dělení	//	Vrátí celou část po dělení	14//3 = 4

Proměnné

- Proměnná
  - označení pro **identifikátor (symbolické jméno)**, které uchovává nějakou informaci (**hodnotu**) určitého **datového typu** při běhu programu
  - jméno proměnné je cesta pro získání reference k uložené hodnotě v paměti počítače
  - zatímco jméno proměnné, typ a lokace jsou často neměnná (pevná), tak data uložená v proměnné mohou být změněna za chodu programu
  - v Pythonu jsou proměnné brány jako odkazy na objekty, díky čemuž je můžeme používat ve funkcích a volat na nich metody
- Typová kontrola proměnné
  - **Statická** typová kontrola (Java): vyžaduje deklarovat proměnnou a definovat její typ, tento typ je dále neměnný
  - **Dynamická** typová kontrola (Python, PHP): typ má hodnota (ne proměnná), proměnnou deklarujeme prvním použitím, do té samé proměnné můžeme ukládat různé typy dat
- Jméno proměnné má určitá omezení: jenom písmena, čísla (ne na začátku) a podrtžítko
- Python rozlišuje velikost písmen u proměnných (`a` a `A` jsou různé proměnné)
- Typ proměnné zjistíme pomocí funkce `type()`

Základní datové typy v Pythonu

- Numerické typy
  - celé číslo `int` x = 5
  - desetinné číslo `float` .25
  - komplexní číslo `complex` 5 + 4j
- Textový typ
  - řetězec `str` “ahoj”
- Sekvenční typy (kolekce dat)
  - seznam `list` [“Marek”, “Hanka”, “Jana”]
  - n-tice `tuple` (“Marek”, “Hanka”, “Jana”)
  - rozsah `range` range(6)
- Mapovací typ
  - slovník `dict` {“jmeno”: “Marek”, “vek”: 36}
- Množinový typ
  - množina `set` {“Marek”, “Hanka”, “Jana”}
- Logický (booleovský) typ
  - logická hodnota `bool` True False

Přiraďovací operace v Pythonu

Operátor	Příklad	Co dělá
=	x = 3	x = 3
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x*3
/=	x /= 3	x = x/3
%=	x %= 3	x = x%3
**=	x **= 3	x = x**3
//=	x //= 3	x = x//3

Hello, World! aneb první program

- Komentář: začíná # a Python dále ignoruje co je za ním na řádce
- Funkce `print()` zobrazí řetězec, číslo nebo hodntu proměnné
- Funkce `input()` přijímá vstup od uživatele jako řetězec, vrací řetězec
- Parsování (casting): specifikuje typ proměnné
  - `int()` vytvoří celé číslo z desetinného (zaokrouhlí dolů) nebo řetězce
  - `float()` vytvoří desetinné číslo z celého čísla nebo řetězce
  - `string()` vytvoří řetězec z jiného datového typu
- Znak `\n` v textu způsobí, že se text za tímto znakem zobrazí na další řádce

Booleovské hodnoty

- Typ `bool` (boolean): pouze dvě hodnoty: `True` a `False`
- Pomocí těchto dvou hodnot se obvykle řídí tok programu
- V Pythonu se jako pravdivý výraz vyhodnotí nenulové číslo, neprázdný řetězec (nebo jiný neprázdný datový typ)
- Booleovskou hodnotu daného typu zjistíme funkcí `bool()`

Porovnávací a logické operátory

Operátor	Zápis
Rovnost	==
Je větší	>
Je menší	<
Je větší nebo rovno	>=
Je menší nebo rovno	<=
Nerovnost	!=

Operátor	Zápis
A zároveň	and
Nebo	or
Negace	not

- operátory `is` a `is not`: porovnávání dvou čísel, objektu představujícího celé číslo a čísla (ale ne dva objekty)
- operátory `in` a `not in`: zjišťuje, zda-li je řetězec obsažený v jiném řetězci (lze použít i s jinými datovými typy)

Podmínka if..else, elif

- Syntaxe: v Pythonu je důležité konzistentní odsazování (oficiální směrnice doporučují velikost zhruba čtyř mezer)
  - Příkaz začíná vždy na novém řádku
  - Blok příkazů je určen odsazením
  - Za výrazy určující tok programu se v Pythonu píše dvojtečka
- Podmínka: výraz, který vrací logickou hodnotu (může a nemusí být v závorkách)
- Když se `if` vyhodnotí jako pravdivý výraz, tak je následující `elif` a `else` konstrukce ignorována (bez ohledu na jeho pravdivost se přeskočí). `Else` a `elif` se provede pouze tehdy, pokud se nesplní podmínka `if`.

Syntaxe if

```
if podmínka:
    blok_příkazů
```

Zkráceně

```
if podmínka: příkaz
```

Syntaxe if..else

```
if podmínka:
    blok_příkazů_1
else:
    blok_příkazů_2
```

Zkráceně

```
příkaz_1 if podmínka else příkaz_2
```

Syntaxe if..elif..else

```
if podmínka_1:
    blok_příkazů_1
elif podmínka_2:
    blok_příkazů_2
elif podmínka_3:
    blok_příkazů_3
else:
    blok_příkazů_4
```

Cykly v Pythonu

- Cyklus `while`
  - nejjednodušší cyklus v Pythonu
  - blok `while` pokračuje v provádění příkazů tak dlouho, dokud platí podmínka
  - pokud podmínka neplatí, provede se nepovinný blok `else` (neprovede se ale v případě, že se z cyklu vyskočí např. příkazem `break`)
- Cyklus `for...in`
  - vykonává příkazy pro hodnoty proměnné v iterovatelném objektu (řetězec, `range`, ...)
- Příkaz `continue`: skočí na začátek bloku příkazů
- Příkaz `break`: vyskočí z cyklu
- Příkaz `pass`: nic nedělá (cykly nemůžou být prádné)
- Funkce `range(od,do,krok)`:
  - range(n): vráti čísla od 0 do n-1
  - range(m,n): vráti čísla od m do n-1
  - range(m,n,i): vrátí čísla od m a každé i-te číslo do n-1

Syntaxe while

```
while podmínka:
    blok_příkazů_1
else:
    blok_příkazů_2
```

Syntaxe for...in

```
for proměnná in iterovatelný_objekt:
    blok_příkazů_1
else:
    blok_příkazů_2
```

Seznamy v Pythonu

- Seznam (list)
  - jeden ze čtyř datových typů v Pythonu umožňujících ukládání kolekcí dat
  - vytváří se za pomoci hranatých závorek, jednotlivé prvky jsou odděleny čárkou
  - vlastnosti prvků seznamu: uspořádané, měnitelné, možnost duplikátů
- Seznamy můžeme vytvářet i za pomoci funkcí `list()` z iterovatelných objektů, např. `range()`
- Indexování
  - k jednotlivým prvkům seznamu lze přistupovat pomocí indexů (0, 1, 2, ...) přes hranaté závorky: `nty_prvek = seznam[n]`
  - v Pythonu můžeme přistupovat k prvkům i odzadu, slouží k tomu záporné indexy (-1, -2, ...)

Indexy	0	1	2	3	4	5	6	7
Seznam	15	3	5	21	8	12	5	3
Záporné indexy	-8	-7	-6	-5	-4	-3	-2	-1

Seznamy v Pythonu

- Ořezávání
  - seznam[m] - vybere jediný prvek
  - seznam[m:n] - vybere prvky v rozsahu m až n-1
  - seznam[m:n:i] - vybere každý i-tý prvek od m do n-1
  - libovolné z m, n, i jde vynechat
  - m, n, i jde zadat i záporné
- Funkce `len()` vrací počet prvků seznamu (iterovatelného objektu)
- Za pomoci indexu můžu prvky měnit: `seznam[index] = novy_prvek`
- Pomocí `in` můžeme otestovat přítomnost prvku v seznamu: `prvek in seznam`
- Procházení seznamu
  - cyklem `for..in` - prvky nemůžeme v procházeném seznamu upravovat
  - pomocí cyklu `for..in` a funkce `range()`
  - pomocí `enumerate()` - vrací index prvku i jeho hodnotu

Seznamy v Pythonu, jejich základní metody

- Metody
  - funkce, které patří nějakému objektu
  - syntaxe: `název_objektu.název_metody(parametry)`
  - metoda typicky změní seznam, na který se použije
  - `seznam.append(prvek)`: připojí prvek na konec seznamu
  - `seznam.insert(i,prvek)`: vloží prvek na i-tou pozici
  - `seznam.pop(i)`: vrátí a odstraní i-ty prvek ze seznamu
  - `seznam.sort()`: seřadí prvky v iterovatelném objektu
  - `seznam.copy()`: vytvoří kopii seznamu (pomocí = vytvoříme jenom referenci na seznam, změny se udělají automaticky v obou seznamech)

Seznamy v Pythonu, jejich základní funkce

- Funkce
  - vstupují do ní parametry a vrací nějakou hodnotu
  - typicky nemění původní seznam
  - `len()`: vrátí počet prvků iterovatelného objektu
  - `min()`: vrátí z iterovatelného objektu prvek s nejmenší hodnotou
  - `max()`: vrátí z iterovatelného objektu prvek s největší hodnotou
  - `sum()`: vrátí součet prvků v iterovatelném objektu
  - `sorted()`: vrací seřazenou kopii seznamu
  - `all()`: vrátí True, jestliže se všechny prvky v iterovatelném objektu vyhodnotí na True, jinak vrátí False
  - `any()`: vrátí True, jestliže se alespoň jeden prvek v iterovatelném objektu vyhodnotí na True, jinak vrátí False
  - `del(seznam[i])`: zmaže i-ty prvek ze seznamu

Textové řetězce v Pythonu

- Textový řetězec (string)
  - jediný textový datový typ v Pythonu
  - vytvoří se uzavřením textu mezi ‘ ’ nebo “ ”, více řádků mezi “’ “’
- Speciální znaky
  - znaky, které nemůžou být v řetězci, se vkládají pomocí escape znaku \
  - \”, \’, \\, `\n` nová řádka, `\t` tabulátor, `\a` zvonek
  - holý řetězec: bere každý znak v řetězci zvlášť, začíná `r` před úvozovkami: `r“ ”`
- Řetězec je iterovatelný objekt
  - lze použít ve funkci `list()` na vytvoření seznamu
  - pomocí `in` můžeme testovat, jestli obsahuje jiný řetězec
  - dá se procházet cyklem `for..in`
  - délka lze zjistit funkcí `len()`

Textové řetězce v Pythonu

- řetězce se dají sčítat (+) a násobit číslem (*)
- převod znaků na ASCII kód pomocí funkce `ord()` a zpět pomocí funkce `chr()`
- řetězce lze porovnávat pomocí standatrních porovnávacích operátorů <, >, >=, <=, ==, !=
  - porovnává se podle abecedy (resp. podle hodnoty v ASCII)
- na jednotlivé znaky se lze dotazovat pomocí indexů jako u seznamů
  - text[m] - vybere jediný znak
  - text[m:n] - vybere znaky v rozsahu m až n-1
  - text[m:n:i] - vybere každý i-tý znak od m do n-1
  - libovolné z m, n, i jde vynechat
  - m, n, i jde zadat i záporné
- nelze ale změnit znak přiřazením `text[i]="znak"` narozdíl od seznamů

Textové řetězce v Pythonu, jejich metody

- Metody
  - `count()`: Vrátí počet podřetězců v jiném řetězci, parametrem je hledaný podřetězec. Lze zadat i výřez.
  - `endswith()/startswith()`: Vrátí True (False) jestli řetězec končí (nekončí)/začíná (nezačíná) zadaným řetězcem, který předáváme jako parametr. Lze zadat i výřez.
  - `find()`: Vrátí index nejlevější pozice podřetězce v jiném řetězci. Hledaný podřetězec předáváme jako parametr. Pokud není nalezen, vrátí -1 . Lze zadat i výřez.
  - `index()`: Dělá v podstatě to samé jako metoda find(), avšak pokud podřetězec nenalezne, vyvolá výjimku.
  - `isalpha()`: Vrátí True/False, pokud jsou všechny znaky v řetězci písmenné znaky a řetězec obsahuje minimálně jeden znak
  - `isdigit()` vrátí True/False, pokud jsou všechny znaky v řetězci číselné znaky (0 - 9) a řetězec obsahuje minimálně jeden znak
  - `islower()/ isupper()`: Vrátí True (False), pokud jsou (nejsou) všechny znaky v řetězci malá/velká písmena a řetězec obsahuje minimálně jeden znak
  - `lower()/upper()`: Převede znaky v řetězci na malá/velká písmena.
  - `replace()`: Nahradí v řetězci podřetězec jiným podřetězcem a vrátí výsledek jako nový řetězec. Parametry metody jsou hledaný podřetězec a podřetězec, kterým jej chceme nahradit.
  - `format()`: Řetězce se dají formátovat pomocí metody format(). Jako argumenty dáme do metody jednotlivé řetězce, nebo odkazy na řetězce. První argument má index 0, další má index 1, atd.

Funkce v Pythonu

- funkcionální styl programování: rozdělíme program na menší části (podproblémy), které řešíme samostatně
- funkce (function)
  - blok kódu, který se vykoná, když je zavolán
  - může přijímat argumenty - data, která zpracuje
  - může vracet hodnotu - výsledek
- argumenty
  - poziční: `arg` na jejich pozici se dosadí argument na stejné pozici při volání funkce
  - klíčové: `kwarg = hodnota` mají dánu výchozí hodnotu a nemusí být při volání funkce zadány v tom pořadí, v jakém jsou deklarovány
  - libovolný počet argumentů: `*args,**kwargs` vloží se do n-tice a můžeme je procházet `for..in`
- rekurze: funkce, která volá samu sebe

Syntaxe funkce (poziční arg.)

```
def funkce(arg1,arg2):
    blok_příkazů
    return: hodnota
```

Syntaxe funkce (klíčové arg.)

```
def funkce(arg1,kwarg2=n):
    blok_příkazů
    return: hodnota
```

Syntaxe funkce (libovolný počet arg.)

```
def funkce(*args):
    blok_příkazů
    return: hodnota
```

Knihovna math

- Knihovny (moduly)
  - poskytují užitečný zdroj datových typů, funkcí a různých nástrojů, pro ještě lepší program
  - import (zpřístupnění) modulu: `import nazev_modulu`
  - funkci z modulu voláme pak: `nazev_modulu.nazev_funkce()`
  - pokud chceme jen určitou funkci: `from nazev_modulu import nazev_funkce`, pak stačí napsat jen: `nazev_funkce()`
  - lze použít i aliasy s následující syntaxí: `import nazev_modulu as vlastni_nazev`
- Knihovna `math`
  - obsahuje pí a Eulerovo číslo
  - goniometrické funkce: `sin(), cos(), tan(), asin(), acos(), atan()`
  - převod radiánů na stupně: `degrees()`
  - zaokrouhlení nahoru: `ceil()` a dolů: `floor()`
  - absolutí hodnota desetinného: `fabs()` a celého: `abs()` čísla
  - `factorial()`
  - druhá mocnina: `pow()` a odmocnina: `sqrt()`
  - logaritmus: přirozený: `log()`, o základu 10: `log10()`, o základu 2: `log2()`
  - více: `help(math)`

Knihovna random

- Knihovna `random`
  - funkce umožňující generovat pseudo-náhodná čísla
  - `randint()`: vybere náhodné číslo v rozsahu od prvního do druhého čísla:
  - `choice()`: vybere náhodný prvek se seznamu (nebo jiné kolekce)
  - `shuffle()`: zamíchá pořadí prvků v seznamu, lze použít i jiný datový typ, který podporuje
  - `sample()`: náhodně vybere množství prvků ze seznamu (nebo jiné kolekce)

Dobrá praxe programátora

- To, že program dělá to, co má, ještě neznamená, že je napsán dobře
  - Není cílem naprogramovat jenom funkční program, ale i přehledný a efektivní
- Pojmenování proměnných
  - anglicky, nebo česky (bez diakritiky), ale nemíchat
  - podle toho, co je v ní uloženo (text, odpoved, soucet,...)
  - typicky: indexy i,j, celé čísla m, n, n, poloměr r, obsah S, objem V
  - ne zkratky (datnar, celsou), ne pom, pomocna nebo dummy
  - víceslovné názvy ano, v Pythonu ve snake_case (datum_narozeni, ciferny_soucet)
- Ošetření vstupu od uživatele
- Syntaxe vs. algoritmus
  - Cílem této přednášky bylo naučit vás základy programování
  - Použitý byl Python, ale základní konstrukce jazyka jsou přenositelné

Easter eggs

- skryté a zábavné funkčnosti, které do jazyka schovali jeho tvůrci:
  - import __*hello*__
  - import *this*
  - import *antigravity*
  - from __future__ import *braces*
- dále si můžete v interaktivní konzoli upravit: '>>>' a '... ' pomocí:

```
>>> import sys
>>> sys.ps1 = "--> "
--> sys.ps2 = "--- " #následuje příklad
--> if True:
---     print("Hello World!")
---
--> Hello World!
```

- Název „Python“ nemá nic do činění s hadem...

Děkuji za pozornost

Případné připomínky, podněty, nebo opravy dejte prosím vědět na email

maria.soltesova@mff.cuni.cz
(maria.soltesova@gmail.com)