Základy algoritmizace

2. Proměnné, datové typy

doc. Ing. Jiří Vokřínek, Ph.D.
Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Základy algoritmizace

- Dnes:
  - Základní práce s proměnnými
  - Datové typy
  - Přesnost a stabilita výpočtu
  - Definice funkcí

Proměnné a základní datové typy

Source: http://core0.staticworld.net/images/idge/imported/article/itw/2013/02/25/numbers_0-100521393-orig.jpg

Proměnné a datové typy

- Při návrhu algoritmu abstrahujeme od binární podoby počítače
- S daty pracujeme jako s hodnotami různých datových typů, které jsou uloženy v paměti předepsaným způsobem

- Datový typ specifikuje
  - **Množinu** hodnot, které je možné v počítači uložit

    Záleží na způsobu reprezentace
  - Množinu operací, které lze s hodnotami provádět

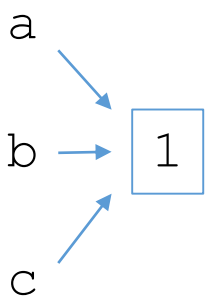
Proměnné a datové typy

- Python nemá explicitní deklaraci typu proměnné

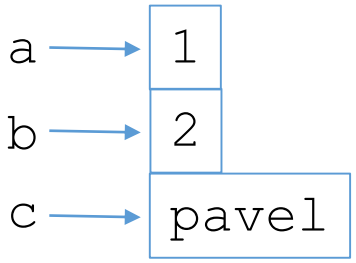
```
number      = 100     #integer
volume      = 10.5    #floating point
name        = "Petr"  #string
```

- Rovnítko je přiřazení (porovnej ==)!
- Vícenásobné přiřazení

```
a = b = c = 1
```




```
a, b, c = 1, 2, "pavel"
```

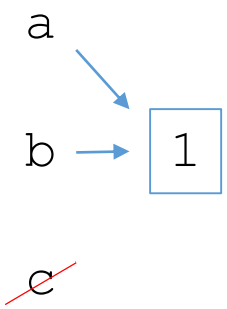


Proměnné a datové typy

- Vymazání proměnné

```
a = b = c = 1

del c
```



- Standartní datové typy
  - Čísla – Numbers
  - Řetězce – String
  - Seznamy – List
  - n-tice – Tuple
  - Slovníky – Dictionary

Proměnné a datové typy

- Číselné typy
  - int – celá čísla se znaménkem (32/64 bit, Python 3 neomezené)
  - long – „velká“ celá čísla (neomezená délka, Python 3 není)
  - float – reálná čísla s plovoucí desetinou čárkou (double – 64bit)
  - complex – komplexní čísla

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFB AEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

Více na http://www.tutorialspoint.com/python/python_numbers.htm

Proměnné a datové typy

- Typ proměnné nedeklarujeme – Python sám pozná co vkládáme
- Vynucená konverze
  - Explicitní zadání hodnot

10L ≈ 10 ≈ 10.0 ≈ "10"

  - Konverzní funkce

    int(x)

    ~~long(x)~~

    float(x)

    complex(x)

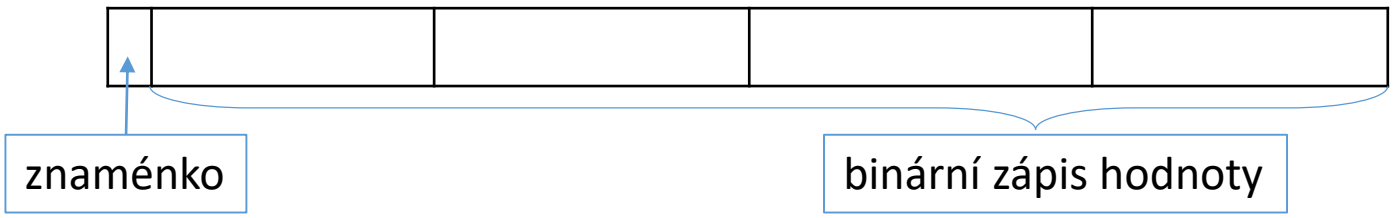
    str(x)

Při konverzi textů je užitečná funkce str.isdigit()

Reprezentace dat

Reprezentace dat

- Integer (pro 32 bit = 4 byte)



  - Znaménkový typ – zakódováno v 1. bitu, zbytek 31 bitů číslo
    - Největší hodnota je 011…111 ≈ 2^{31}-1 = 2147483647

000…000 je 0!

    - Nejmenší hodnota je 100…000 ≈ -2^{31} = -2147483648

0 je již v kladných

  - Pro záporná čísla je použit tzv. doplňkový kód

  - **big vs. little endian** – pořadí uložení bytů (platí pro všechny datové typy)

Platí obecně!

Reprezentace dat

- Integer – v Python 3 neomezená délka

```
a = 15
b = 123456789101112131415
a.bit_length()                    4
b.bit_length()                    67
```

Zkuste si!

Reprezentace dat

- Float (norma IEEE 754 – 64 bit = 8 byte double)

exponent (*exp*)	mantisa (*m*)

  - Aproximace daná rozsahem paměťového místa
  - Reálné číslo x je reprezentováno jako $\mathrm{x} = m \cdot z^{exp}$
  - Mantisa je normalizována $0{,}1 \leq m < 1$
  - Exponent i mantisa jsou uloženy jako celá čísla

Reprezentace dat

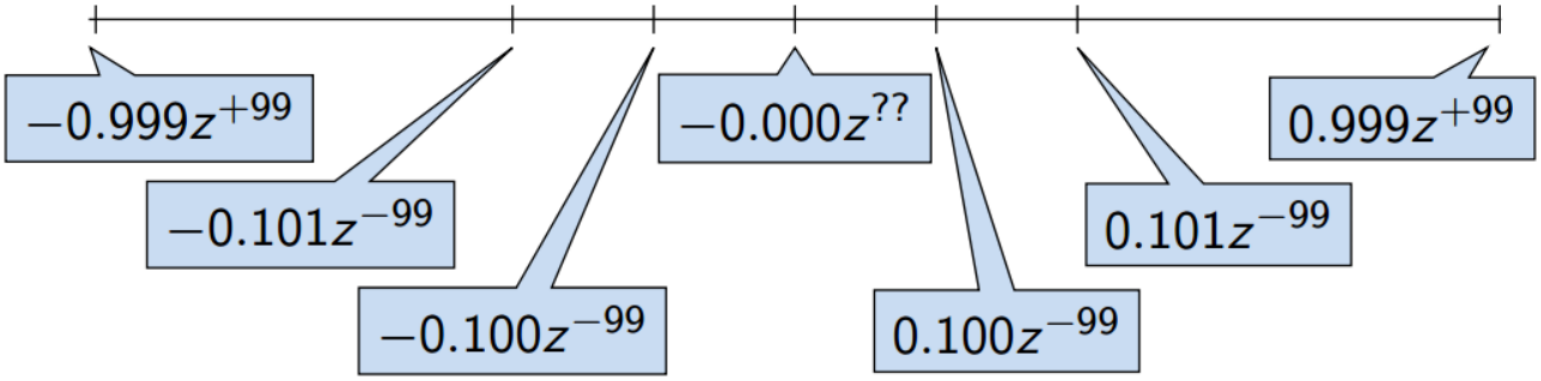
- Příklad
  - 7 byte
  - Mantisa 3 byte + znaménko
  - Exponent 2 byte + znaménko
  - Základ z = 10
  - Nula

- $x = 77{,}5 = 0{,}775 \cdot z^{+02}$

Reprezentace dat

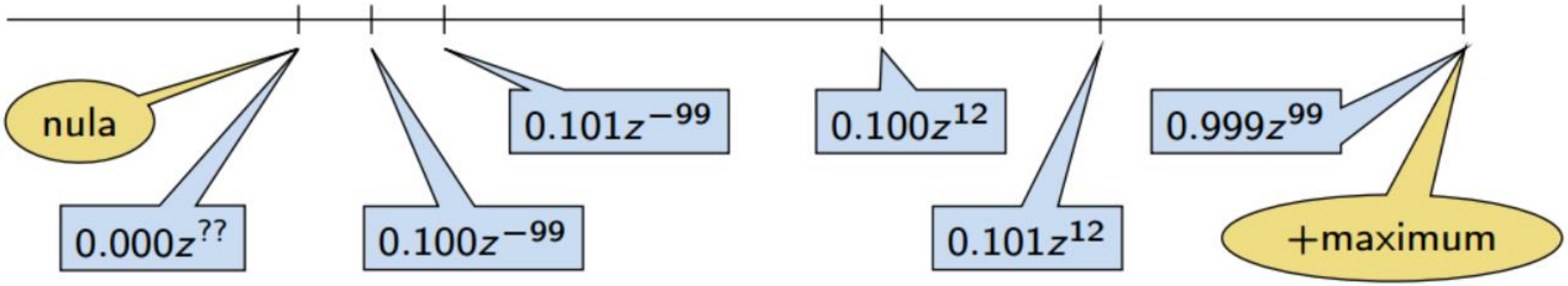
- Příklad
  - Maximální kladné číslo $+0{,}999 \cdot z^{+99}$
  - Minimální kladné číslo $+0{,}100 \cdot z^{-99}$
  - Maximální záporné číslo $-0{,}100 \cdot z^{-99}$
  - Minimální záporné číslo $-0{,}999 \cdot z^{+99}$

+	99	+	999
-	99	+	100
-	99	-	100
+	99	-	999



Reprezentace dat

- Příklad
  - Rozsah hodnot pro konkrétní exponent je dán velikostí mantisy
  - Absolutní vzdálenost dvou aproximací tak záleží na exponentu
    - Mezi hodnotou 0 a 1,0 je využit celý rozsah mantisy pro každý exponent



  - Aproximace reálných čísel nejsou na číselné ose rovnoměrně!

Reprezentace dat

- Double dle IEEE 754
  - 64 bit (8 byte)
  - 1 bit znaménko s (+ nebo –)
  - 11 bitů exponent, tj. 2048 možností
  - 52 bitů mantisa, ≈ 4.5 biliardy možností

    4 503 599 627 370 496
    - Neumožňuje přesně uložit čísla se zápisem delším než 52 bitů
    - Čím větší exponent, tím větší „mezery“ mezi sousedními aproximacemi čísel
  - Reálné číslo *x* se zobrazuje ve tvaru

$$x = (-1)^{S} \cdot \text{mantisa} \cdot 2^{\text{exponent} - \text{bias}}$$

  - bias umožňuje reprezentovat exponent vždy jako kladné číslo

    Např. bias = $2^{eb-1} - 1$, kde eb je počet bitů exponentu, tj. bias = 1023

Přesnost výpočtu

Source: http://s1216.photobucket.com/user/TheThingy/media/photo-1.png.html

Přesnost výpočtu

- Příklad – zápis čísla $\frac{1}{3}$ v dekadické soustavě
  - $= 0{,}33333 \ldots 3333$
  - $= 0{,}3\overline{3}$
  - $\approx 0{,}33333333333$
  - $\approx 0{,}333$

V trojkové soustavě lze vyjádřit jako 0,1 $(0 \cdot 3^1 + 0 \cdot 3^0 + 1 \cdot 3^{-1})$

- Nepřesnosti zobrazení reálných čísel v konečné posloupnosti bitů způsobují
  - Iracionální čísla, např. $e, \pi, \sqrt{2}$
  - Čísla, která mají v dané soustavě periodický rozvoj, např. $\frac{1}{3}$
  - Čísla, která mají příliš dlouhý zápis

… dvojková soustava

Přesnost výpočtu

- Ztráta přesnosti při aritmetických operacích
  - Sčítání

```
a = 1e+10
b = 1e-10
print(a)                    10000000000.0
print(b)                    1e-10
print(a+b)                  10000000000.0
print(a+b-a)                0.0
```

Přesnost výpočtu

- Ztráta přesnosti při aritmetických operacích
  - Dělení

```
number = 1000000
v = 0
for i in range(0,number):
    v += 1/10
print(v)
```

100000.0000133288

```
number = 10000000
v = 0
for i in range(0,number):
    v += 1/10
print(v)
```

999999.9998389754

Pozor, Python může „podvádět“ ve výpisech!

Přesnost výpočtu

- Ztráta přesnosti při aritmetických operacích
  - Dělení

```
number = 5
v = 0
check = number * 0.1
for i in range(0,number):
    v += 0.1
print(v)
print(check)
print(check==v)
```

0.5
0.5
True

```
number = 6
v = 0
check = number * 0.1
for i in range(0,number):
    v += 0.1
print(v)
print(check)
print(check==v)
```

0.6
0.6000000000000001
False

"True" pro 1, 2, 3, 4, 5, 13, 14, 44, 45, 190, 191, 192, 751, 752, 753, 3012, 3013, 3014, 12044, 12045, 48190, 48191 (do 100000)

Přesnost výpočtu

- Příklady
  - Přesná reprezentace 0,1

```
.1 + .1 + .1 == .3
```

False

0.1000000000000000055511151231257827021181583404541015625

  - Zaokrouhlení

```
round(2.675, 2) == 2.67
```

True

2.674999999999998223643160599749535322189331054687 5

  - Více na

https://docs.python.org/3.5/tutorial/floatingpoint.html

Přesnost výpočtu

- Strojová přesnost ε_m – nejmenší desetinné číslo, které přičtením k 1.0 dává výsledek různý od 1,

  pro $|v| < \varepsilon_m$ platí $\quad v + 1.0 == 1.0$

- Zaokrouhlovací chyba – nejméně ε_m

- Přesnost výpočtu – aditivní chyba roste s počtem operací v řádu $\sqrt{N} \cdot \varepsilon_m$

  Často se však kumuluje v jednom směru v řádu $N \cdot \varepsilon_m$

Přesnost výpočtu

- Zdroje chyb
  - Chyby matematického modelu – mat. aproximace fyz. situace
  - Chyby vstupních dat
  - Chyby numerické metody
  - Chyby zaokrouhlovací

  - Absolutní chyba aproximace

    $\mathrm{E}(\mathrm{x}) = \hat{x} - x,\ \hat{x}$ je přesná hodnota, x aproximace
  - Relativní chyba $\mathrm{RE}(\mathrm{x}) = \frac{\hat{x}-x}{x}$

Přesnost výpočtu

- Podmíněnost numerických úloh
  - Podmíněnost úlohy $C_p = \frac{relativní\ chyba\ výtupních\ údajů}{relativní\ chyba\ vstupních\ údajů}$
  - Dobře podmíněná úloha $C_p \approx 1$
  - Výpočet je dobře podmíněný, je-li málo citlivý na poruchy ve vstupních datech
  - Numericky stabilní výpočet – vliv zaokrouhlovacích chyb na výsledek je malý
  - Výpočet je stabilní, je-li dobře podmíněný a numericky stabilní

Přesnost výpočtu

- Příklady numerických chyb
  - Ariane 5 – 4.6.1996

    Datová konverze z 64-bit desetinné reprezentace na 16-bit znaménkový integer

    40 sekund po startu explodovala

    http://www.esa.int/esaCP/Pr_33_1996_p_EN
  - Systém Patriot – 25.2.1991

    Systémový čas v desetinách vteřiny, počítán přičítáním 1/10

    Po 100 hodinách provozu chyba 0,34 vteřiny (≈půl kilometru letu rakety Scud)

    28 mrtvých, 98 zraněných

    http://www.ima.umn.edu/~arnold//disasters/patriot.html

Proměnné a datové typy

- Standartní datové typy
  - Čísla – Numbers ✓
    - Matematické funkce
    - Trigonometrické funkce
    - Náhodná čísla
    - Konstanty (e, pi)

http://www.tutorialspoint.com/python/python_numbers.htm

  - Řetězce – String
  - Seznamy – List
  - n-tice – Tuple
  - Slovníky – Dictionary

http://www.tutorialspoint.com/python/python_variable_types.htm

Proměnné a datové typy

- String
  - Řetězec znaků v uvozovkách
  - Přístup k podřetězcům pomocí [] a [:] (od nuly!)
  - Spojování a opakování pomocí + a *
  - Test obsahu in a not in

```
str = 'Hello World!'
print(str)                  Hello World!
print(str[0])               H
print(str[2:5])             llo
print(str[2:])              llo World!
print(str * 2)              Hello World!Hello World!
print(str + "TEST„)         Hello World!TEST
print("H" in str)           True
print("ell" not in str)     False
```

  - Formátování a další funkce

http://www.tutorialspoint.com/python/python_strings.htm

Proměnné a datové typy

- List
  - Seznam uzavřený v [] oddělený čárkami
  - Může obsahovat různé typy dat
  - Přístup k členům pomocí [] a [:] (od nuly!)
  - Spojování a opakování pomocí + a *
  - Test obsahu in a not in

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']
print(list)                          ['abcd', 786, 2.23, 'john', 70.2]
print(list[0])                       abcd
print(list[1:3])                     [786, 2.23]
print(list[2:])                      [2.23, 'john', 70.2]
print(tinylist * 2)                  [123, 'john', 123, 'john']
print(list + tinylist)               ['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
print("john" not in list)            False
```

Proměnné a datové typy

- List
  - Můžeme mazat
  - Může měnit obsah
  - Počet členů – funkce len
  - Další funkce a operace

http://www.tutorialspoint.com/python/python_lists.htm

```
list = ['physics', 'chemistry', 1997, 2000];

print(list)                 ['physics', 'chemistry', 1997, 2000]
print(len(list))            4
print(list[2])              1997
list[2] = 2001
print(list[2])              2001
del list[2]
print(list)                 ['physics', 'chemistry', 2000]
print(len(list))            3
```

Proměnné a datové typy

- Tuple
  - Seznam uzavřený v () oddělený čárkami
  - Podobně jako seznamy ale jen pro čtení
  - „Immutable“
  - Funkce a operace

http://www.tutorialspoint.com/python/python_lists.htm

Proměnné a datové typy

- Dictionary
  - Asociativní pole – adresář, slovník … hash table
  - Obsahuje páry klíč-hodnota v {}
  - Přístup k hodnotám přes klíče v []

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
```

```
print(dict['one'])          This is one
print(dict[2])              This is two
print(tinydict)             {'dept': 'sales', 'code': 6734, 'name': 'john'}
print(tinydict.keys())      ['dept', 'code', 'name']
print(tinydict.values())    ['sales', 6734, 'john']
```

Proměnné a datové typy

- Dictionary
  - Přidávání a aktualizace elementů
  - Mazání elementů pomocí del nebo funkce clear

    del dict['Name']

    dict.clear()

  - Vlastnosti klíčů
    - Bez duplikací (přepisují se)
    - Klíče musí být „immutable“ – čísla, řetězce, n-tice

  - Užitečné funkce

    dict.get(key, default=None)

    dict.has_key(key)

    http://www.tutorialspoint.com/python/python_dictionary.htm

Proměnné a datové typy

- Standartní datové typy
  - Čísla – Numbers ✓
    - Matematické funkce
    - Trigonometrické funkce
    - Náhodná čísla
    - Konstanty (e, pi)

http://www.tutorialspoint.com/python/python_numbers.htm

  - Řetězce – String ✓
  - Seznamy – List ✓
  - n-tice – Tuple ✓
  - Slovníky – Dictionary ✓

http://www.tutorialspoint.com/python/python_variable_types.htm

Funkce

Funkce

- Znovupoužitelný blok kódu
- Provádí jednu konkrétní funkci
- Podporuje modularitu programované aplikace
- Přijímá množinu argumentů
- Může vracet výsledek

```
def printme( parameters ):
   print(parameters)
   return

printme("Hello!")
```

Funkce

- Argumenty jsou vždy předávány referencí!
- Cokoli změníme v těle funkce, má vliv i mimo

```
def changeme( mylist ):
    mylist.append([1,2,3,4])
    print(mylist)
    return

list = [10,20,30]
changeme(list)
```

[10, 20, 30, [1, 2, 3, 4]]

- Pokud ale změníme referenci, nepřenese se mimo funkci

```
def changeme( mylist ):
    mylist = [1,2,3,4]
    print(mylist)
    return

list = [10,20,30]
changeme(list)
print(list)
```

[1, 2, 3, 4]

[10, 20, 30]

Funkce

- Funkce mají omezený „scope“ proměnných
- Proměnné definované v rámci funkce mají pouze lokální scope
- Proměnné definované mimo funkce mají globální scope (jsou přístupné odevšad)
- Vzájemně se překrývají, ale neovlivňují

```
def test():
    scope = "local"
    print(scope)
    return

scope = "global"
print(scope)
test()
print(scope)
```

global
local
global

Funkce

- Funkce mají omezený „scope“ proměnných
- Proměnné definované v rámci funkce mají pouze lokální scope
- Proměnné definované mimo funkce mají globální scope (jsou přístupné odevšad)
- Vzájemně se překrývají, ale neovlivňují

```
def test(scope):
    scope = "local"
    print(scope)
    return scope

scope = "global"
print(scope)
scope = test(scope)
print(scope)
```

```
global
local
local
```

Funkce

- Pojmenované argumenty
  - Nezáleží na pořadí
- Defaultní argumenty
  - Nemusí být uvedeny
  - Nesmí za ním být žádný non-default!

```
def doSomething( first , second = "none" ):
    print(first + second)
    return;

doSomething( second=2, first=1 )                3
doSomething( second="two", first="one" )        onetwo
doSomething("some")                             somenone
```

Funkce

- Proměnné počty parametrů (*tuple)
  - Formální argumenty jsou následovány proměnnou s hvězdičkou
  - Pokud nejsou předány, seznam je prázdný

```
def doSomething( *args):
    print(len(args))
    return;

doSomething( 1,2,3,4,5 )                5
doSomething()                           0
```

- Anonymní funkce
  - Nepoužívá def
  - Jednořádková funkce, vrací jen vyhodnocení výrazu
  - Nepracuje s globálními proměnými

http://www.tutorialspoint.com/python/python_functions.htm

Funkce

- Praktická doporučení
  - Funkce by měla být krátká – dělat jen jednu věc
  - Malý počet argumentů
  - Jméno volíme jako sloveso, př. computeFactorial(n)
  - Název funkce i argumentů by měl být samo-vypovídající
  - Snažíme se vyvarovat přepínání činnosti funkce hodnotou vstupních parametrů

Funkce

- Dnes:
  - Základní práce s proměnnými
  - Datové typy
  - Přesnost a stabilita výpočtu
  - Definice funkcí

Příště řízení běhu programu