Základy algoritmizace

1. Úvod

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

Základy algoritmizace

- O čem je předmět?
  - Naučit se myslet algoritmicky

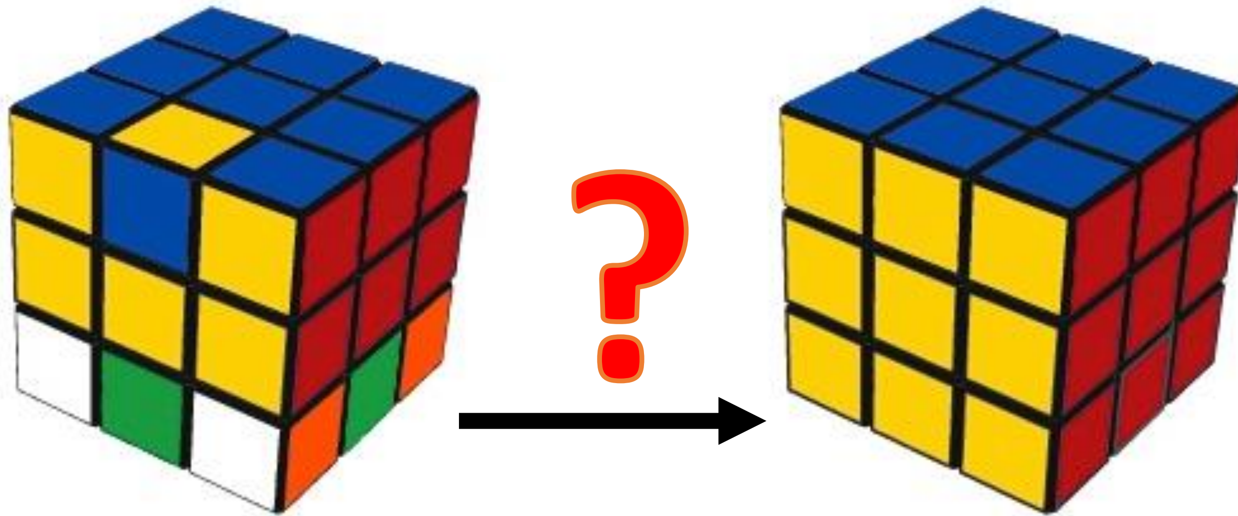
    Algoritmus řeší problém
  - Naučit se ovládat počítač – programovat v Pythonu (trochu)

    Program řídí počítač
  - Získat přehled o základních algoritmech a práci s daty

    Jak funguje moje navigace?

Source: http://www.marcinkossakowski.com/wp-content/uploads/2014/11/dijkstra-map.jpg

Základy algoritmizace



Algoritmus!
→ program

Source: http://www.qedcat.com/rubiks_cube/

Základy algoritmizace

- Dnes:
  - Program
  - Začínáme s Pythonem
  - Proměnné a data
  - Přesnost výpočtu
  - Výjimky

Program a programovací jazyk

Source: http://codehackersblog.blogspot.com/2015/06/dont-call-yourself-programmer.html

Výpočetní prostředky (počítače)

- Jednoúčelové přístroje s předepsaným chováním
  - program / posloupnost kroků (instrukcí) je vestavěná a neměnná

Kalkulačka, pračka, první telefony

- Počítač s uloženým programem v paměti
  - Posloupnost instrukcí čtena z paměti
  - Flexibilita ve tvorbě posloupnosti

Program lze libovolně měnit

  - Architektura počítače se společnou pamětí pro data a program
    - Von Neumannova architektura počítače

John Louis von Neumann (1903–1957)

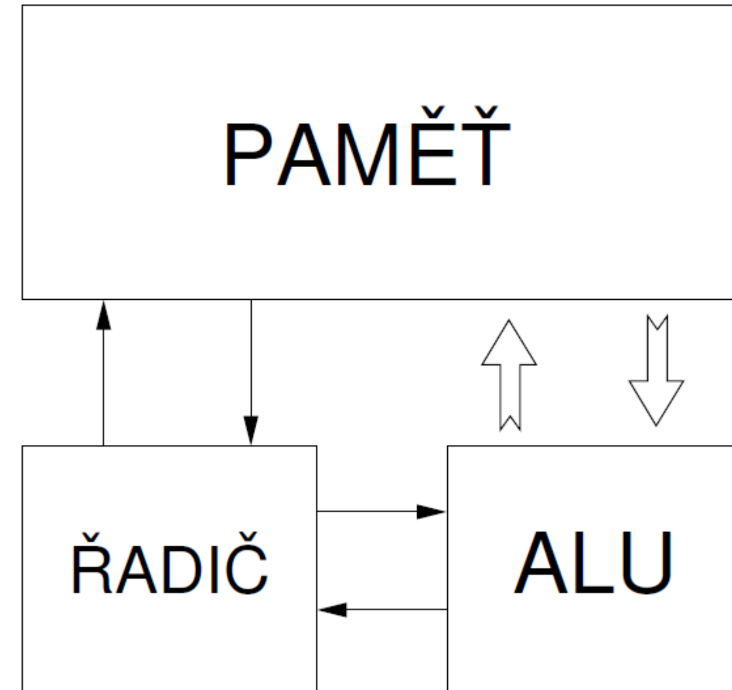
Von Neumannova architektura

- ALU – Aritmeticko-logická jednotka (Arithmetic Logic Unit)

  Základní matematické a logické instrukce

- PC – Čítač instrukcí (Program Counter)

  „Ukazuje" na místo v paměti s instrukcemi pro vykonání



V drtivé většině případů je program posloupnost instrukcí zpracovávající jednu nebo dvě hodnoty (uložené v nějakém paměťovém místě) jako vstup a generování nějaké výstupní hodnoty, kterou ukládá někam do paměti nebo modifikuje hodnotu PC (podmíněné řízení běhu programu).

Assembler

- Jazyk symbolických adres
- Nízko-úrovňový (nad strojovým kódem)

```
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32  PROC             ; procedure begins here
       CMP  AX,97       ; compare AX to 97
       JL   DONE        ; if less, jump to DONE
       CMP  AX,122      ; compare AX to 122
       JG   DONE        ; if greater, jump to DONE
       SUB  AX,32       ; subtract 32 from AX
DONE:  RET              ; return to main program
SUB32  ENDP             ; procedure ends here
```

Source: http://www.cybercomputing.co.uk/Languages/Languages/Low_level/assembly.html

Program je „recept“

- Program je posloupnost kroků (výpočtů) popisující průběh výpočtu pro řešení problému

(je to „recept” na řešení problému)

- Pro zápis receptu potřebujeme jazyk

Způsob zápisu programu

- Jazyk definuje základní sadu primitiv (operací/příkazu), které můžeme použít pro zápis receptu
- S konečnou množinou primitiv dobrý programátor naprogramuje „cokoliv”.

Co muže být vyjádřeno

- Turing Machine – obecný model počítacího stroje

Alan Turing, 1936

- V předmětu B6B36ZAL používáme programovací jazyk Python

Programování není o znalosti konkrétního programovacího jazyka, je to o způsobu uvažování a řešení problému

Správnost programu

- Syntakticky i staticky sémanticky správný program neznamená, že dělá to co od něj požadujeme
- Správnost a smysluplnost programu je dána očekávaným chováním při řešení požadovaného problému
- V zásadě při spuštění programu mohou nastat tyto události:
  - Program havaruje a dojde k chybovému výpisu

    Mrzuté, ale výpis (report) je dobrý start řešení chyby (bug)
  - Program běží, ale nezastaví se a počítá v nekonečné smyčce.

    Zpravidla velmi obtížné detekovat a program ukončujeme po nějaké době
  - Program včas dává odpověď

    Je však dobré vědět, že odpověď je korektní

Správnost programu je plně v režii programátora, proto je důležité pro snadnější ověření správnosti, ladění a hledání chyby používat **dobrý programovací styl**

Program a jeho zápis

- Program – popis činnosti prováděné počítačem
- Programovací jazyk – notační systém pro zápis programu

Program zpravidla zapisujeme ve zdrojových (textových) souborech

- Čitelnost programu:
  - strojová – efektivnost kódu
  - lidská – srozumitelnost, udržovatelnost, kódovací konvence,

Vývojová prostředí (editor), debugger, nástroje pro správu verzí, analýza, testování → ***softwarové inženýrství***

- Abstrakce
  - datová – základní typy, struktury, modulární
  - řídící – základní, strukturální, modulární

Jak vypadá program?

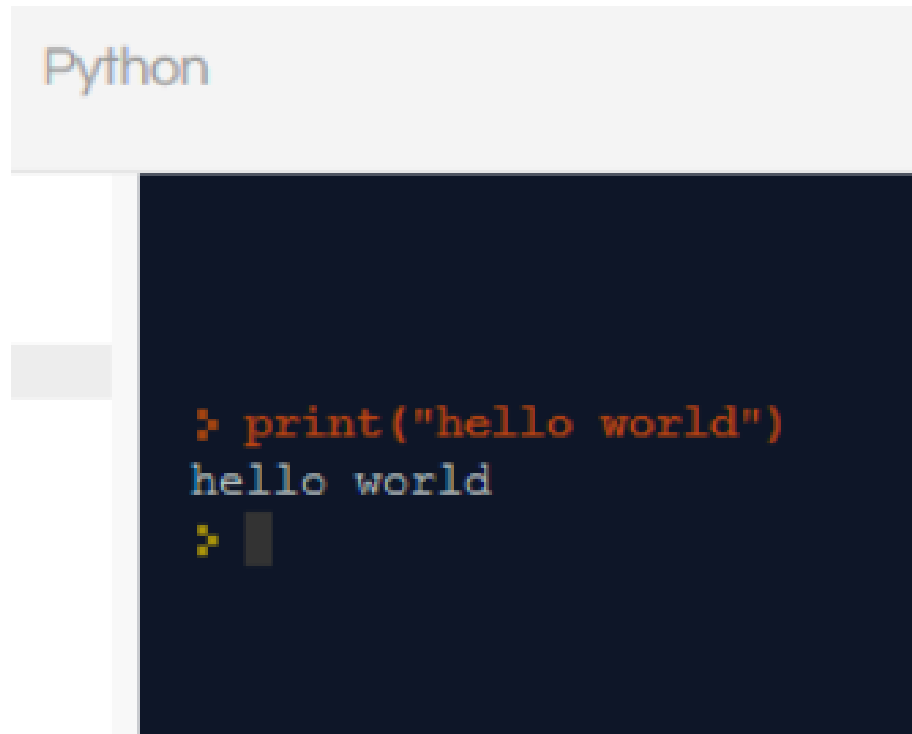
```
    ...r_user(self, user):
        Returns a QuerySet of connections for user.
        """
        set1 = self.filter(from_user=user).select_rela
        set2 = self.filter(to_user=user).select_relatec
        return set1 | set2
    def are_connected(self, user1, user2):
        if self.filter(from_user=user1, to_user=user2).c
            return True
        if self.filter(from_user=user2, to_user=user1).cc
            return True
        return False
    def remove(self, user1, user2):
        """
        Deletes proper object regardless of the order of u
        """
        connection = self.filter(from_user=user1, to_user=u
        if not connection:
            connection = self.filter(from_user=user2, to_use
        connection.delete()
--:---  models.py      Top L1      (Python AC
```

Source: http://learntocodewith.me/wp-content/uploads/2014/04/python-language.jpg?230531

Příklad

- Hello world – nejčastější „první program“

  > print(“hello world”)



Příklady na https://cw.fel.cvut.cz/wiki/courses/pri-bootcamp/01

Výstup

print("Hello World!")	`Hello World!`
print(100)	`100`
print(3+6)	`9`
print(3*6)	`18`
print(6*2-2*4)	`4`
print(2*(3+6))	`18`

```
print("3*6")
print(3*6)
```

```
3*6
18
```

```
print("abc"+"def")
print("abc"+123)
```

```
abcdef
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Komenář

```
# toto je komentar

print("Hello World!") # toto je komentar

"# toto neni komentar"
```

Vstup

- Textový řetězec

```
input("zadej text")
```

Pro zpracování lze uložit do proměnné,

případně přetypovat – viz. příští přednáška

```
vstup = input("Jak se mas?")
print("Ja se mam taky "+vstup)
```

Zkuste si

- Hra s geniální umělou inteligencí

```
print("Je to "+input("Zadej cislo, ktere mam hadat "))
```

Proměnné a základní datové typy

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

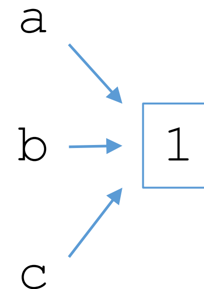
Proměnné a datové typy

- Python nemá explicitní deklaraci typu proměnné

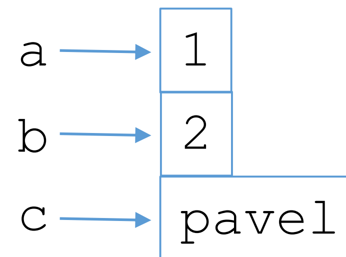
```
number      = 100     #integer
volume      = 10.5    #floating point
name        = "Petr"  #string
```

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

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




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

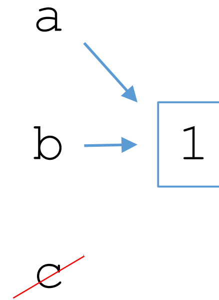


Proměnné a datové typy

- 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)

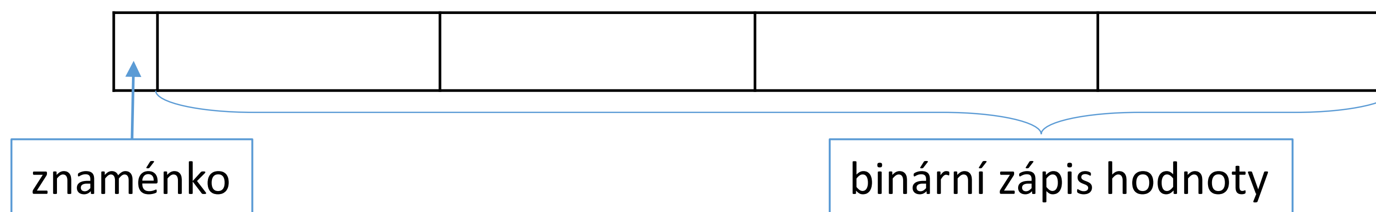
    str(x)

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

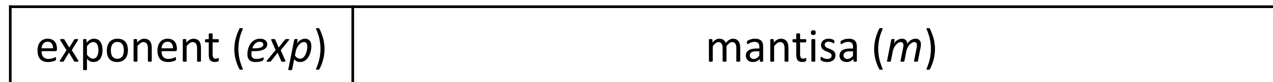
- Integer – v Python 3 neomezená délka

```
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)



  - 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

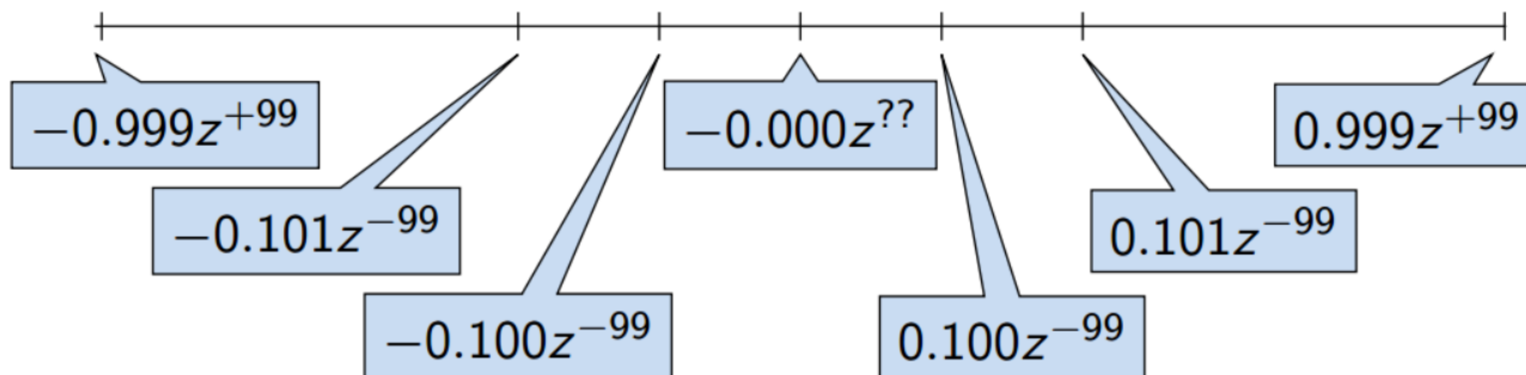
- Příklad
  - 7 byte
  - Mantisa 3 byte + znaménko
  - Exponent 2 byte + znaménko
  - Základ z = 10
  - Nula

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

Reprezentace dat

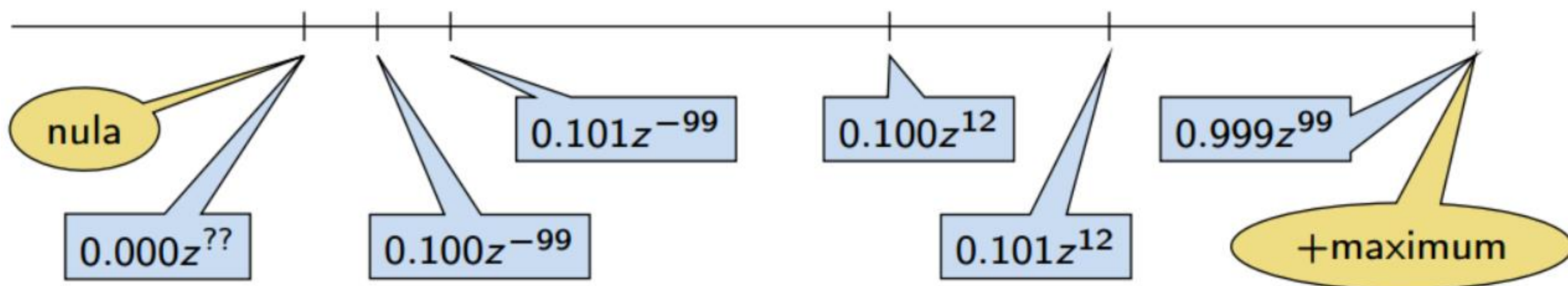
- Příklad
  - Maximální kladné číslo $+0,999 \cdot z^{+99}$
  - Minimální kladné číslo $+0,100 \cdot z^{-99}$
  - Maximální záporné číslo $-0,100 \cdot z^{-99}$
  - Minimální záporné číslo $-0,999 \cdot z^{+99}$

+	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
  - $= 0{,}33333 \ldots 3333$
  - $= 0{,}33\overline{3}$
  - $\approx 0{,}333333333333$
  - $\approx 0{,}333$

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)
print(b)
print(a+b)
print(a+b-a)
```

```
10000000000.0
1e-10
10000000000.0
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

2.67499999999999982236431605997495353221893310546875

  - 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 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

- 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'])
print(dict[2])
print(tinydict)
print(tinydict.keys())
print(tinydict.values())
```

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['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

Výjimky

Výjimky

- K řízení výjimečných stavů programu
- Událost, která vzniká při běhu programu a přeruší jeho *normální* tok
- Výjimka musí být *ošetřena*, nebo program končí
- Příklad výjimky
  - ZeroDivisionError, IndexError, IOError, TypeError

```
a= 1 + "1"
```

```
Traceback (most recent call last):
  File „program.py", line 1, in <module>
    a= 1 + "1"
TypeError: unsupported operand type(s) for +: 'int' and 'str'

Process finished with exit code 1
```

Výjimky

- Zachycení a ošetření výjimek

```
try:
    block of operations
except Exception1:
    handle Exception1 type here
except Exception2:
    handle Exception2 type here
else:
    execute this block in case of no exception
```

- Příklad

```
try:
    a= 1 + "1"
except:
    print("Something wrong")
else:
    print("Everything is OK")
```

Výjimky

- Zachycení více výjimek

```
try:
    block of operations
except(Exception1[, Exception2[,...ExceptionN]]]):
    if there is any exception from given list
    then execute this block
else:
    if there is no exception execute this block
```

- try-finally
  - Nezachytí výjimku, není ovlivněno except a else
  - Blok finally se vždy provede

```
try:
    block of operations
    this will be skipped due to any exception
finally:
    this would be allways executed
```

Pokud nastala výjimka, „vyvolá“ se po bloku `finally`

Výjimky

- try-finally
  - Příklad práce se souborem

```
try:
    fh = open("testfile", "w")
    try:
        fh.write("This is my test file.")
    finally:
        print("Going to close the file")
        fh.close()
except IOError:
    print("Error: can\'t find file or read data")
```

Výjimky

- Výjimky s argumenty
  - Umožňují předat informaci o důvodu chyby
  - Jedna hodnota nebo tuple

(error string, error number, error location)

```
try:
   You do your operations here
except ExceptionType as argument:
   You can print value of argument here
```

- Příklad

```
def getIntVal(var):
   try:
      return int(var)
   except ValueError as argument:
      print("No numbers in\n", argument)

getIntVal("xyz")
```

Výjimky

- Vyvolávání výjimek

```
raise Exception("my exception")

Traceback (most recent call last):
  File "program.py", line 1, in <module>
    raise Exception("my exception")
Exception: my exception

Process finished with exit code 1
```

- Po zachycení lze opět vyvolat

```
try:
    a=1+"1"
except:
    print("Something wrong")
    raise
```

Výjimky

- Obecná pravidla
  - Zachytáváme jen to, co umíme napravit
  - Napravujeme na správném místě a správným způsobem
  - Nepoužíváme pro ošetření očekávaných stavů

    výjimka = něco výjimečného
  - Vyvarujeme se „tichému“ zachycení

```
try:
    a=1+"1"
except:
    pass #TODO: something wrong?
```

Základy algoritmizace

- Dnes:
  - Program
  - Začínáme s Pythonem
  - Proměnné a data
  - Přesnost výpočtu
  - Výjimky

Příště problémy, algoritmy, data

Source: http://scifi.stackexchange.com/questions/19782/how-does-seeing-the-code-help-neo