# 1. Úvod do programování

## BAB37ZPR – Základy programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

# Přehled témat

# Část I

## O předmětu

# I. O předmětu

Organizace předmětu

Studijní výsledky

# Předmět a lidé

- Webové stránky předmětu

  https://cw.fel.cvut.cz/wiki/courses/bab37zpr
- Přednášející a garant předmětu
  - Stanislav Vítek, viteks@fel.cvut.cz

    http://mmtg.fel.cvut.cz/personal/vitek/
- Cvičící
  - Václav Vencovský, vencovac@fel.cvut.cz
  - Adam Zizien, zizieada@fel.cvut.cz
- Konzultace
  - MS Teams – po domluvě

# Cíle předmětu

- **Motivovat k programování**
  - Programování je klíčová dovednost, která může hrát rozhodující roli na trhu práce
- **Naučit se algoritmizovat**
  - Formulace problému a návrh řešení
  - Rozklad problému na dílčí úlohy
  - Identifikace opakujících se vzorů
- **Získat zkušenosti s programováním**
  - Základní programovací konstrukce

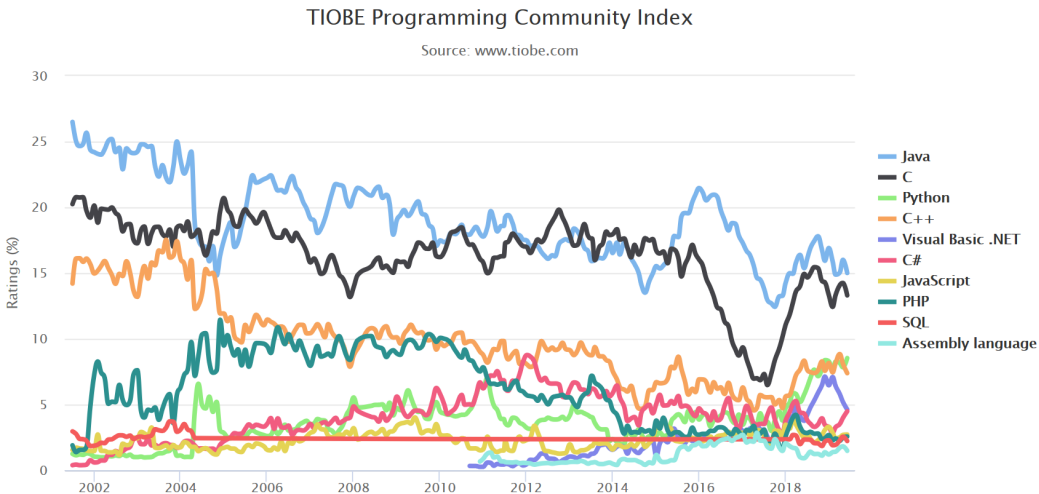
    Proměnné, cykly, podmínky, datové struktury a jednodušší algoritmy
  - Programovací jazyk Python, řada principů obecně použitelných

    Cvičení, domácí úkoly, hledání chyb, práce s dokumentací, test

Programátorovi nestačí perfektní znalost programovacího jazyka, ale především musí vědět, jak vůbec danou úlohu řešit.

# Proč Python?




TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
Java
C
Python
C++
Visual Basic .NET
C#
JavaScript
PHP
SQL
Assembly language

# Proč Python? 2/2

- jazyk vysoké úrovně, všeobecné použití, dobře čitelný
- velmi populární, mnoho knihoven, multiparadigmatický
- dynamický, interpretovaný (byte-code)
- s automatickou alokací paměti

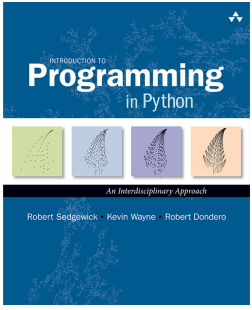
# Organizace a hodnocení předmětu

- **Studijní výsledky**
  - Průběžná práce v semestru – domácí úkoly a test
  - Zkouškový a implementační test
- **Docházka**
  - Přednášky jsou nepovinné, ale snad přínosné a zábavné
  - Cvičení jsou povinná, možné dvě omluvené absence

    V případě distanční výuky není vyžadována online účast, k dispozici bude audiovizuální záznam cvičení.
  - Na cvičení se očekává aktivní účast při řešení příkladů

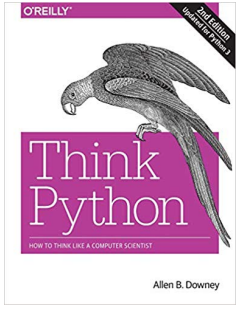
    Na cvičení je třeba se **připravit**, nejlépe návštěvou přednášky a studiem podkladů (řešené příklady)
- **Řešení problémů**
  - Obracejte se na svého cvičícího
  - Při komunikaci e-mailem pište vždy ze své fakultní adresy
  - Do předmětu zprávy uvádějte zkratku předmětu ZPR
  - V případě zásadních problémů uvádějte do CC též přednášejícího

# Zdroje a literatura



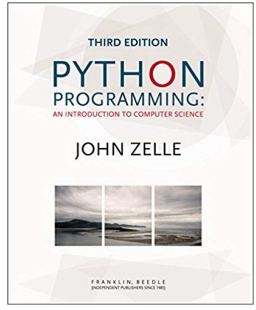

Robert Sedgewick
Introduction to Programming in Python: An Interdisciplinary Approach
Addison-Wesley
2015

ISBN 978-0134076430




Allen B. Downey
Think Python: How to Think Like a Computer Scientist
O'Reilly Media
2015

ISBN 978-1491939369




John Zelle
Python Programming: An Introduction to Computer Science
Franklin, Beedle & Associates
2016

ISBN 978-1590282755

# I. O předmětu

# Domácí úkoly

- Samostatná práce s cílem osvojit si praktické zkušenosti s programováním
- Jednotné zadání na prednášce a jednotný termín odevzdání
- Náročnost domácích úkolů se postupně zvyšuje
- Odevzdání domácích úkolů prostřednictvím systému BRUTE
- Cílem řešení úkolů je získat **vlastní** zkušenost
  - Neopisujte – škodíte především sobě
  - Provádíme automatickou kontrolu plagiátů u všech odevzdaných řešení
    - každý s každým
    - každý s řešením z minulých let (pokud je podobný příklad)
    - u podezřelých případů provedeme manuální kontrolu
  - V případě odhalení jsou potrestání **oba** účastníci incidentu

Pokud nečemu nerozumíte, ptejte se!

# Hodnocení

| Zdroj bodů | Maximum | Nutné minimum |
|---|---|---|
| Domácí úkoly | 50 | 20 |
| Test v semestru | 10 | 0 |
| Semestrální práce | 20 | 10 |
| Zkouška | 20 | 10 |
| Součet | 100 | 40 |

- Úkoly mají stanovené deadlines, pozdní odevzdání je penalizováno.
- Domácí úkoly musí být odevzdány nejpozději do 8.1.2023 ve 23:59 CET!
- Test v semestru – test na počítači, teoretické otázky (zveřejněny na CW)
- Zkouška – implementace několika příkladů na počítači, cca 4 hodiny

# Klasifikace

| Klasifikace | Bodové rozmezí | Slovní hodnocení |
|---|---|---|
| A | $\geq 90$ | výborně |
| B | $80 - 89$ | velmi dobře |
| C | $70 - 79$ | dobře |
| D | $60 - 69$ | uspokojivě |
| E | $50 - 59$ | dostatečně |
| F | $< 50$ | nedostatečně |

# Přehled přednášek

| | | |
|---|---|---|
| 1. | Informace o předmětu, úvod do programování. Primitivní datové typy. | 19.9. |
| 2. | Řidící struktury, větvení a cykly. Funkce | 26.9. |
| 3. | Příklady a jednoduché algoritmy. | 3.10. |
| 4. | Složené datové typy. | 10.10. |
| 5. | Řazení a třídění, složitost. | 17.10. |
| 6. | Složitější algoritmy, rekurze. | 24.10. |
| 7. | Objekty. | 31.10. |
| 8. | Abstraktní datový typ, zásobník, fronta, spojový seznam. | 7.11. |
| 9. | Stromové struktur, rozptylová tabulka. | 14.11. |
| 10. | Konečný automat, regulární výrazy. | 21.11. |
| 11. | Aplikace – numerické výpočty, vizualizace. | 28.11. |
| 12. | Aplikace – signal processing. | 5.12. |
| 13. | Aplikace – zpracování číselných řad, machine learning. | 12.12. |
| 14. | Zkouškový test – předtermín. | 9.1. |

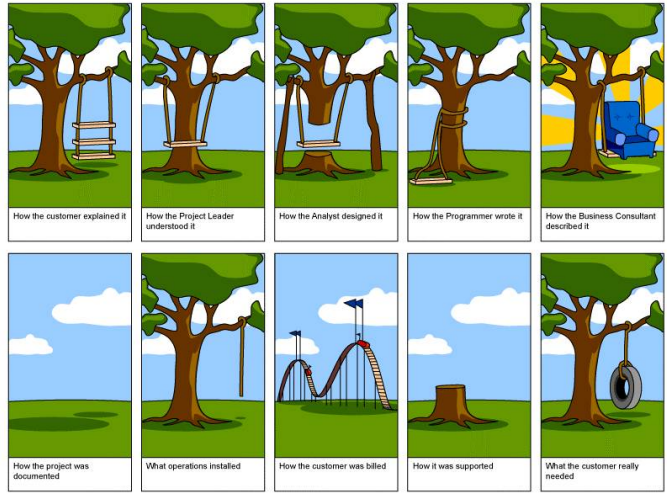
# Část II

## O programování

# II. O programování

# Řešení problémů

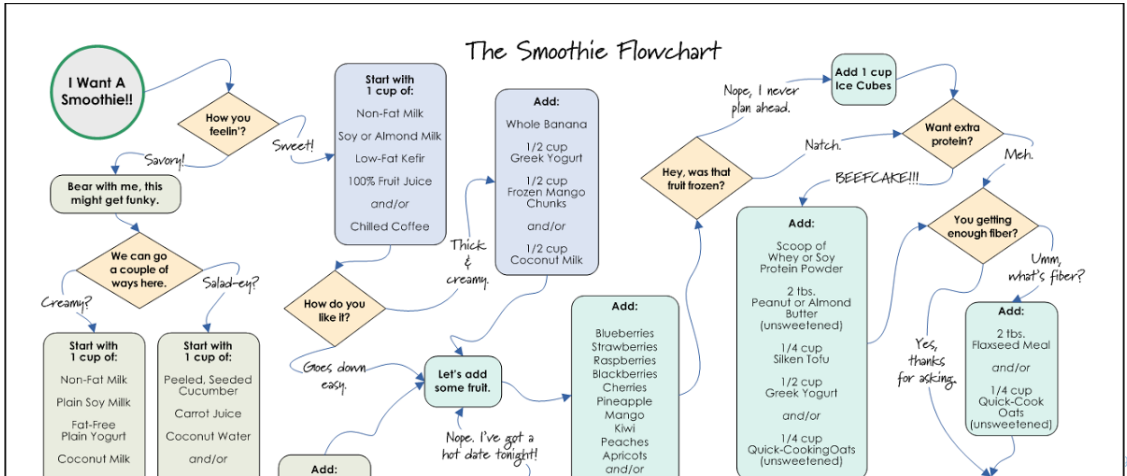
1. formulace problému
2. analýza možných řešení
3. návrh algoritmu
4. implementace
5. ověření funkčnosti
6. optimalizace
7. oprava chyb
8. údržba
9. dokumentace



# Co je to program?

- Program je **recept** – posloupnost kroků (výpočtů), popisující průběh řešení nějakého problému pomocí dostupných prostředků – programovací prostředí, počítač, ...

Receptu budeme říkat algoritmus.



# Co je to algoritmus?

- Návod nebo postup, jak provést určitou činnost.
- Algoritmus by měl být tak podrobný, aby mu porozumněl i počítač.
- Vlastnosti algoritmu:
  1. Skládá se z konečného počtu jednoduchých činností – kroků.
  2. Po každém kroku lze určit, jak se má pokračovat nebo skončit.
  3. Počet opakování jednotlivých kroků algoritmu je vždy konečný.
  4. Vede ke správnému výsledku.
  5. Algoritmus lze použít k řešení celé (velké) skupiny podobných úloh.

**Manželka:** kup chleba a když budou mít rohlíky, vezmi jich deset.
**Manžel, programátor:** přinese z obchodu deset chlebů, protože rohlíky měli.

https://www.youtube.com/watch?v=Ct-l0OUqmyY

Slovo algoritmus vzniklo odvozením od jména perského matematika Al-Chorezmího, jehož jméno bylo ve středověku latinizováno jako Al-Gorizmí.

# Zápis algoritmu

- Existují 4 hlavní způsoby, jakými lze algoritmus popsat:
  - slovně – vyjádříme slovně postup řešení a jednotlivé kroky
  - graficky – použití vývojových diagramů a struktogramů
  - matematicky – jednoznačnný popis matematickou konstrukcí (např. rovnicí)
  - programem – kroky algoritmu jsou popsány programovacím jazykem
- Návrhy algoritmů:
  - shora dolů – problém rozdělíme na několik podúloh, které řešíme
  - zdola nahoru – z triviálních úloh skládáme vyšší úlohy
  - kombinace obou metod

V praxi vždy záleží především na komplexnosti a povaze řešeného algoritmus, který postup bude nejlepší aplikovat.

# Základní složky programů a algoritmů

- Programy zpravidla transformuje množinu vstupních dat na množinu dat výstupních
- Základní složky programů

  Vstup dat – načtení dat programem, interaktivní nebo ze souboru dat
  Popis dat – volba datového typu a umístění v paměti
  Zpracování – výpočet definovaný algoritmem, řízení toku programu
  Výstup – interakce s uživatelem nebo např. zápis do souboru

- Řízení toku programu

  Posloupnost – jeden nebo několik kroků, které se provedou právě jednou v daném pořadí
  Cyklus – opakování nějaké posloupnosti, dokud je splněna podmínka opakování
  Větvení – volba posloupnosti instrukcí na základě vyhodnocení podmínky

- Kombinace základních složek algoritmu umožňuje vytvářet komplexní konstrukce.
- Pokud se některé části algoritmu opakují, je vhodné poslupnosti orgranizovat do větších celků: procedur a funkcí (podprogramů).

# Jak začít?

## Jednoduché algoritmy, grafické programování

- Scratch – MIT Media Lab
- Angry Birds
- Code with Anna and Elsa
- Minecraft

## Programovací jazyk Karel

- Pohyb robota po čtvercové sítí
- Richard E. Pattis, Karel The Robot: A Gentle Introduction to the Art of Programming, Stanford, 1981
- Online: Stanford , Oldřich Jedlička

Další zajímavé programovací jazyky pro výuku programování např. zde .

# A co Python?

- Pokud máme Python nainstalovaný (a systémová cesta `PATH` odkazuje do adresáře, kde je umístěn binární soubor), stačí ho spustit příkazem `python`
- Pokud Python nainstalovaný nemáme, tak si ho nainstalujeme :-)
  - `https://www.python.org/downloads`
    - včetně editoru IDLE
    - instalační nástroj `pip`
  - `https://www.anaconda.com/`
    - včetně IDE Spyder
    - instalační nástroj `conda` a správa prostředí pomocí Anaconda Navigator
  - `https://jupyter.org/`
    - interaktivní prostředí
    - běží na lokálním počítači, editor v prohlížeči
- Pokud si Python (zatím) instalovat nechceme
  - `https://colab.research.google.com`
    - online interaktivní prostředí
    - velké množství předinstalovaných modulů

# II. O programování

# Python jako kalkulačka – interaktivní mód

```
>>> 3+8
11
>>> 11*(5+3)
88
>>> 128./16.
8.0
>>> 2**16
65536
```

- Píšeme **výrazy**, které obsahují
  - Celá čísla: `3`, `8`, ...
  - Reálná čísla: `128.`, `11.5`,...
  - Operátory: `+`, `-`, `/`, `*`,
  - Oddělovače: `(`, `)`
- Co se děje v zákulisí?
  - Spustili jsme program `python3`, **interpret** Pythonu
  - Opakované vykonávání (smyčka, *loop*)
    - tisk výzvy *(prompt)* `>>>`
    - přečtení uživatelského vstupu *(read)*
    - vyhodnocení výrazu *(evaluate)*
    - tisk výsledku *(print)*

## Poznámka k syntaxi

- Python je **case sensitive** – velikost písmen je důležitá
- Diakritika – Python3 umožňuje používat UTF kódování, raději to ale dělat nebudeme
- Komentáře – symbol `#`

# Program jako transformace

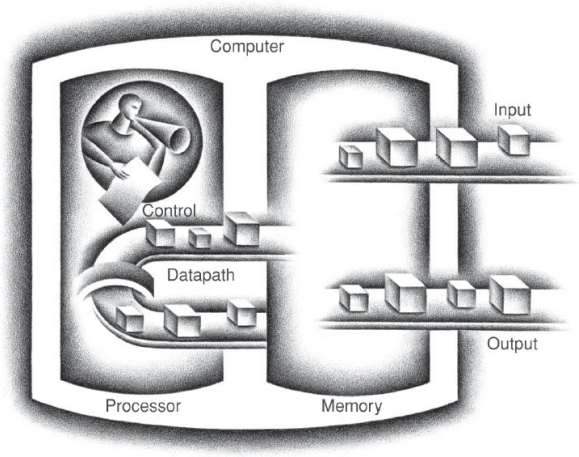



Figure: Základní komponenty počítače (Zdroj: Patterson, D. A.; Hennessy, J. L. Computer organization and design ARM edition: the hardware software interface. Morgan kaufmann, 2016.)

# II. O programování

# Proměnné a přiřazení

- Hodnotu výrazu lze uložit pro pozdější použití:

```
>>> a = 3          # identifikátor = výraz
>>> b = 3 + a
```

- Jaká je hodnota proměnné **b**?

```
>>> b
6
```

- Proměnná udržuje hodnotu, která se může měnit (proto proměnná)
- Proměnná má datový typ:
  - primitivní: číslo (`int`, `float`), pravdivostní hodnota (`bool`)
  - strukturovaný: řetězec (`string`), seznam (`list`) / pole (`array`), slovník (`dict`)
- Vícenásobná přiřazení

```
>>> x, y, z = "Orange", "Banana", "Cherry"
>>> x = y = z = "Orange"
```

# Názvy proměnných

- Posloupnosti písmen, čísel a znaků '_', první musí být písmeno

```
>>> 2myvar = "John"
  File "<stdin>", line 1
    2myvar = "John"
    ^
SyntaxError: invalid decimal literal
```

- Názvy by měly jasně vysvětlovat, jakou hodnotu popisují
- Víceslovné názvy proměnných:
  - **CamelCase**

    ```
    >>> myVariableName = "John"
    ```
  - **PixedCase**

    ```
    >>> MyVariableName = "John"
    ```
  - **SnakeCase**

    ```
    >>> my_variable_name = "John"
    ```
- Nelze používat klíčová slova Pythonu (`if`, `else`, `True`, ...)

# Proměnné – příklad 1

```
>>> boys=15
>>> girls=17
>>> total=boys+girls
>>> difference=girls-boys
>>> ratio=boys/total
>>> total
32
>>> difference
2
>>> ratio
0.46875
```

# Proměnné – příklad 2

```
>>> # hodnoty proměnných lze měnit
>>> a=10
>>> a=a-2
>>> a=a*2
>>> # jaká je hodnota a?
>>> a
16
```

## Proč používat proměnné

- **DRY** = Do not repeat yourself – šetřme si práci, neopakujme se
- Zlepšení
  - **srozumitelnosti** – smysluplná jména proměnných
  - **údržby** – jedna změna jen na jednom místě
  - **efektivity** – využijeme předchozích výpočtů

# Výrazy a operace

- **Výrazy** – kombinace proměnných, konstant a volání funkcí pomocí operátorů
- **Operace** – aritmetické, logické, řetězení textových řetezců, ...
- Pořadí vyhodnocování se řídí **prioritou** a **asociativitou**

**Příklad** výrazy a operátory

```
>>> x = 13
>>> y = x % 4 # modulo
>>> y = y + 1
>>> y += 1
>>> a = (x==3) and (y==2)
>>> b = a or not a
>>> s = "petr"
>>> t = "klic"
>>> u = s + t
```

# Přiřazení a rovnost

- přiřazení =

```
>>> # přiřaď do x hodnotu 3
>>> x = 3
```

- test na rovnost ==

```
>>> # otestuj zda v x je hodnota 3
>>> x == 3
True
```

- častá chyba: záměna = a ==

# Operátory

- Většina operátorů intuitivních
  - aritmetické: `*`, `/`, `+`, ...
  - logické: `and`, `or`
- Mírné odlišnosti od jiných jazyků
  - celočíselné dělení: `//`
  - mocnina: `**`
- Zkrácený zápis operátoru: `y += 5` odpovídá `y = y + 5`
- Pořadí vyhodnocování vesměs intuitivní (algebra)
- Pokud jste na pochybách
  - konzultujte dokumentaci
  - závorkujte
- Zkrácené vyhodnocení: `1 + 1 == 2 or x == 3`

# II. O programování

# Datové typy v programovacích jazycích

- Jak jsou typy deklarovány?
  - **explicitně** – zápis programátorem v kódu, např. `int x;`
  - **implicitně** – typ je určen automaticky kompilátorem
- Jak se provádí typová kontrola?
  - **staticky** – na základě kódu (při kompilaci)
  - **dynamicky** – za běhu programu

## A co Python?

- Dynamické implicitní typování – typ se určuje automaticky a může se měnit
- **Deklarace** proměnné – první přiřazení hodnoty
- Zjištění typu: `type`, `isinstance`
- Možnost explicitního uvedení typu při deklaraci a přetypování

# Číselné datové typy

```
>>> # celá čísla - int
>>> a = 5
>>> print(a, "je typ", type(a))
5 je typ <class 'int'>

>>> # reálná čísla - float
>>> a = 2.0
>>> print(a, "je typ", type(a))
2.0 je typ <class 'float'>

>>> # komplexní čísla - complex
>>> a = 1+2j
>>> print(a, "je komplexni cislo?", isinstance(a, complex))
(1+2j) je komplexni cislo? True
```

# Strukturované datové typy

```
>>> # seznam hodnot stejných nebo různých datových typů
>>> a = [1, 2.2, 'python']
>>> print(a, "je typ", type(a))
[1, 2.2, 'python'] je typ <class 'list'>

>>> # uspořádaná n-tice -- tuple
>>> a = (3, 3.14)
>>> print(a, "je typ", type(a))
(3, 3.14) je typ <class 'tuple'>

>>> # slovník -- dict
>>> a = {"brand": "Ford", "model": "Mustang", "year": 1964}
>>> print(a, "je typ", type(a))
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964} je typ <class 'dict'>
```

# Explicitní přetypování

- Pokud existuje jednoznačná konverze mezi hodnotami dvou datových typů, je možné provést přetypování (cast)
- V některých případech může dojít ke změně hodnoty (ztráta přesnosti při konverzi `float`→`int`)

```
>>> x = int(1)        # x -> 1
>>> y = int(2.8)      # y -> 2
>>> z = int("3")      # z -> 3
>>> x = float(1)      # x -> 1.0
>>> y = float(2.8)    # y -> 2.8
>>> z = float("3")    # z -> 3.0
>>> w = float("4.2")  # w -> 4.2
>>> x = str("s1")     # x -> 's1'
>>> y = str(2)        # y -> '2'
>>> z = str(3.0)      # z -> '3.0'
```

# II. O programování

# Standardní výstup – výpis jedné proměnné 1/4

```
>>> # výpis hodnoty proměnné s automatickým odřádkováním
>>> x = 10
>>> print(x)
10

>>> # výpis s doprovodnou informací
>>> print('x = ', x)
x =  10

>>> # formátováni pomocí modifikatorů, Python 2.x
>>> print('x = %d' % x)
x = 10

>>> # f-funkce, Python 3.6 a novejsi
>>> print(f'x = {x}')
x = 10
```

# Standardní výstup – výpis více proměnných 2/4

```
>>> x = 10; y = 11; z = 12
>>> print(x, y, z) # hodnoty automaticky oddeleny ' '
10 11 12
>>> print(x, y, z, sep=';') # potlaceni separatoru ''
10;11;12
>>> print(x, y, z, sep='\n') # odradkovani separatorem
10
11
12
>>> print('x={}, y={}'.format(x,y))
x=10, y=11
>>> print('x={1}, y={0}'.format(x,z))
x=12, y=10
```

# Standardní výstup – formátovací funkce 3/4

```
>>> print('Prvni', 'Druhy', 'Treti', sep=', ', end=', ')
Prvni, Druhy, Treti,
>>> print('Ctvrty', 'Paty', 'Sesty', sep=', ')
Ctvrty, Paty, Sesty
```

- formátovací funkce

  `.rjust`, `.ljust`, `.center`
  `.rstrip`, `.lstrip`

- další možnosti:

  `sys.write()`, ...

# Standardní výstup – výrazy 4/4

- Pozor na rozdílné chování operátorů definovaných pro různé datové typy

```
>>> a = 10; b = 20; c = 30; print(a + b + c)
60
>>> a = "10"; b = "20"; c = "30"; print(a + b + c)
102030
>>> a = 10; b = "20"; print(a + b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> a = "10"; b = 20; print(a + b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> print(a + str(b) + ' ', int(a) + b) # lze podle potřeby přetypovat
1020  30
```

# Formátovaný výstup 1/3

```
>>> # %[flags][width][.precision]type
>>> # integer a float
>>> print("A : % 2d, B : % 5.2f" % (1, 05.333))
A :  1, B :  5.33
>>> # integer
>>> print("A : % 3d, B : % 2d" % (240, 120))
A :  240, B :  120
>>> # octal
>>> print("% 7.3o"% (25))
    031
>>> # float v exponencialnim tvaru
>>> print("% 10.3E"% (356.08977))
 3.561E+02
```

Tento způsob formátování pravděpodobně v další verzi Pythonu nebude.

# Formátovaný výstup 2/3

```
>>> # kombinace pozicniho argumentu a klicoveho slova
>>> print('Na hristi jsou {0}, {1}, a {other}.'
...       .format('Cesi', 'Slovaci', other ='rozhodci'))
Na hristi jsou Cesi, Slovaci, a rozhodci.
>>> # formatovani cisel
>>> print('A :{0:2d}, B :{1:8.2f}'
...       .format(12, 00.546))
A :12, B :    0.55
>>> # zmena pozicniho argumentu
>>> print('B: {1:3d}, A: {0:7.2f}'.format(47.42, 11))
B:  11, A:   47.42

>>> print('A: {a:5d},  B: {p:8.2f}'
...       .format(a = 453, p = 59.058))
A:   453,  B:    59.06
```

# Formátovaný výstup 3/3

```
>>> str = "ahoj"
>>> # tisk centrovaneho retezce a vyplnoveho znaku
>>> print (str.center(40, '#'))
##################ahoj##################
>>> # tisk retezce zarovnaneho vlevo a vyplnoveho znaku
>>> print (str.ljust(40, '-'))
ahoj------------------------------------
>>> # tisk retezce zarovnaneho vpravo a vyplnoveho znaku
>>> print (str.rjust(40, '+'))
++++++++++++++++++++++++++++++++++ahoj
```

# II. O programování

# P1.1 První program

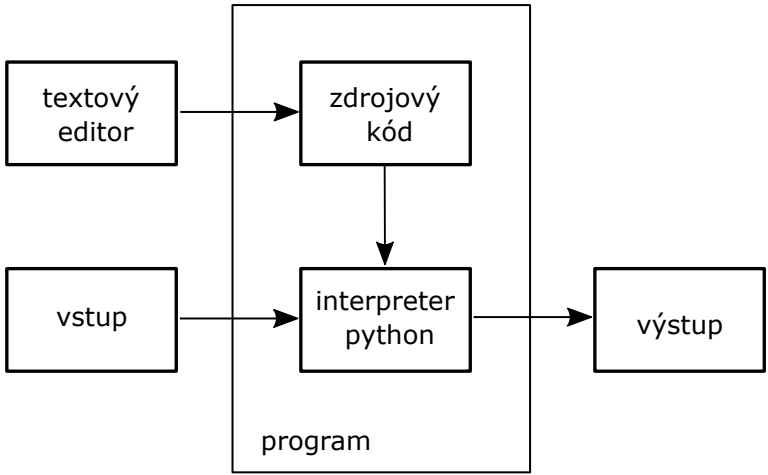
- vytvoříme v textovém editoru
- uložíme do souboru `hello.py`
- spustíme (z příkazové řádky, opakovaně)

```
# Vypise pozdraveni
print("Hello world")
```

lec01/hello.py

```
$ python3 hello.py
```

# Editace a interpretace programu



# II. O programování

# P1.2 Převod jednotek teploty 1/3

- Kolik stupňů Celsia je 75 stupňů Fahrenheita?

```
>>> f=75
>>> c=(f-32)*5./9.
>>> print(c)
23.88888888888889
```

- Trochu hezčí výpis:

```
>>> print(f, " °F je ", c, " °C.")
75  °F je  23.88888888888889  °C.
```

- Další zlepšení:

```
>>> print("%f °F je %f °C." % (f, c))
75.000000 °F je 23.888889 °C.
>>>
>>> print("%0.1f °F je %0.1f °C." % (f, c))
75.0 °F je 23.9 °C.
```

# P1.2 Převod jednotek teploty 2/3

- Co když chceme převést více hodnot?
- Vytvoříme program, který budeme moci opakovaně spouštět!
- V textovém editoru vytvoříme soubor `units.py`

```
# Prevod stupnu Fahrenheita na stupne Celsia
f=75
c=(f-32)*5./9.
print("%0.1f F je %0.1f C." % (f, c))
```

lec01/units.py

```
$ python units.py
75.0 F je 23.9 C.
```

- Program převádí pouze jednu hodnotu.
- Co třeba převádět hodnotu načtenou z příkazové řádky?

# P1.2 Převod jednotek teploty 3/3

- Vylepšená verze s načítáním čísla z příkazového řádku

```
# Prevod stupnu Fahrenheita na stupne Celsia
import sys

f=int(sys.argv[1]) # první argument
c=(f-32)*5./9.
print("%0.1f F je %0.1f C." % (f, c))
```

lec01/units2.py

```
$ python units.py 75
75.0 F je 23.9 C.
$ python units.py 60
60.0 F je 15.6 C.
$ python units.py -20
-20.0 F je -28.9 C.
```