

2. Řídicí struktury

BAB37ZPR – Základy programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Proměnné, datové typy
- Část 2 – Řízení toku výpočtu
- Část 3 – Funkce
- Část 4 – Ladění

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 přetypování

```
>>> x = float(y)
>>> y = str(158)
```

Datové typy v Pythonu

- Číselné – `int`, `float`, `complex`

```
1 | a = 5
2 | print(a, "je typ", type(a))
4 | a = 2.0
5 | print(a, "je typ", type(a))
7 | a = 1+2j
8 | print(a, "je komplexni cislo?", isinstance(a, complex))
```

- Seznam – `list`

```
1 | a = [1, 2.2, 'python']
```

- Textový řetězec – `str`
- Další typy: uspořádaná n-tice `tuple`, množina `set`, slovník `dict`

Standardní výstup 1/3

- výpis proměnné (s automatickým odřádkováním)

```
>>> x = 10
>>> print(x)
10
```

- výpis s doprovodnou informací

```
# standardni zpusob, formatovani pomoci funkci
print('x = ', x)
# spojovani (concatenate) textovych retezcu
print('x = ' + str(x))
# Python 2.x style, formatovani pomoci modifikatoru
print('x = %d' % x)
# f-funkce, Python 3.6 a novejsi
print(f'x = {x}')
```

Standardní výstup 2/3

- tisk více hodnot

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

```
10 11 12
10;11;12
10
11
12
x=10, y=11
x=12, y=10
```

Standardní výstup 3/3

- bez odřádkování

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

```
Prvni, Druhy, Treti, Ctvrty, Paty, Sesty
```

- formátovací funkce

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

- další možnosti:

```
sys.write(), ...
```

Příklad – formátovaný výstup

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

```
A : 1, B : 5.33
A : 240, B : 120
031
3.561E+02
```

Tento způsob formátování pravděpodobně v další verzi Pythonu nebude.

Příklad – formátovaný výstup

```
# kombinace pozicního argumentu a klicového slova
print('Na hristi jsou {0}, {1}, a {other}.'
      .format('Cesi', 'Slovaci', other = 'rozhodci'))
# formatovani cisel
print('A :{0:2d}, B :{1:8.2f}'
      .format(12, 00.546))
# zmena pozicního argumentu
print('B: {1:3d}, A: {0:7.2f}'.format(47.42, 11))
print('A: {a:5d}, B: {p:8.2f}'
      .format(a = 453, p = 59.058))
```

Na hristi jsou Cesi, Slovaci, a rozhodci.

A :12, B : 0.55

B: 11, A: 47.42

A: 453, B: 59.06

Příklad – formátovaný výstup

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

```
#####ahoj#####  
ahoj-----  
+++++++ahoj
```

Část I

Řízení toku výpočtu

I. Řízení toku výpočtu

Větvení

Cykly

Porovnávání (čísel)

- výsledkem operace porovnávání je logická hodnota **True** nebo **False** (typ `bool`)
- Operátory `>`, `<`, `==`, `>=`, `<=`, `!=`

```
>>> 8 > 3
True
>>> 10 <= 10
True
>>> 1==0
False
>>> 2!=3
True
>>> a=4
>>> b=6
>>> a<b
True
```

Podmíněný příkaz `if`

- Umožňuje větvení programu na základě podmínky
- Má tvar:

```
if podminka:  
    prikaz1  
else:  
    prikaz2
```

- `podminka` – logický výraz, jehož hodnota je logického typu
`False` (hodnota 0) nebo `True` (hodnota různá od 0)
- podle toho, jak se provede podmínka, se provede jedna z větví
- `else` větev nepovinná
- možné vícenásobné větvení

Podmíněný příkaz `if`

```
3 import sys
5 n = int(sys.argv[1])
6 # první argument - celé číslo
7 if n>0:
8     print(n, " je kladne cislo.")
10 print("Konec programu.")
```

lec02/conditionals.py

```
$ python conditionals.py 10
10 je kladne cislo.
Konec programu.
$ python conditionals.py -1
Konec programu.
```

- Bloky kódu jsou v Pythonu určeny odsazením.
- Bloky kódu = základ strukturovaného programování.

Větvení if-else

```
3 import sys
5 n = int(sys.argv[1])
6 # první argument
7 if n>0:
8     print(n, " je kladne cislo.")
9 else:
10    print(n, " není kladne cislo.")
```

lec02/conditionals2.py

```
$ python conditionals2.py 10
10 je kladne cislo.
$ python conditionals.py -5
-5 není kladne cislo.
```

Vnořené větvení

```
3 import sys
5 n = int(sys.argv[1])
6 # první argument
7 if n>0:
8     print(n, " je kladne cislo")
9 else:
10     if n==0:
11         print(n, " je nula")
12     else:
13         print(n, " je zaporne cislo")
```

lec02/conditionals3.py

```
$ python conditionals3.py 0
0 je nula.
```

Zřetězené podmínky if-elif-else

```
3 import sys
5 n = int(sys.argv[1])
6 # prvni argument
7 if n>0:
8     print(n, " je kladne cislo")
9 elif n==0:
10    print(n, " je nula")
11 else:
12    print(n, " je zaporne cislo")
```

lec02/conditionals4.py

```
$ python conditionals4.py 0
0 je nula.
```

Příklad – maximum tří čísel

```
3 import sys
5 a=int(sys.argv[1])
6 b=int(sys.argv[2])
7 c=int(sys.argv[3])
9 if a>b: # a nebo c
10     if a>c: # a > b, a > c
11         print(a)
12     else: # c >= a > b
13         print(c)
14 else: # b >= a
15     if b>c: # b > c, b >= a
16         print(b)
17     else: # c >= b >= a
18         print(c)
```

lec02/maxi.py

```
$ python maxi.py 2 3 5
0 je nula.
```

I. Řízení toku výpočtu

Větvení

Cykly

Cyklus `for`, `range`

- Známý počet opakování cyklu:
- Má tvar:

```
for x in range(n):  
    prikaz
```

- provede příkazy pro všechny hodnoty `x` ze zadaného intervalu
- `range(n)` – interval od 0 do `n-1` (tj. `n` opakování)
- `range(a, b)` – interval od `a` do `b-1`
- `for/range` lze použít i obecněji (nejen intervaly) – viz později/samostudium

Příklady

- faktoriál

```
3 | n = int(input())
5 | f = 1
7 | for i in range(1, n+1):
8 |     f = f * i
10 | print(f)
```

lec02/factorial-for.py

- posloupnost

```
1 | for i in range(n):
2 |     print(2**i - 2*i, end=" ")
```

```
1 0 0 2 8 22 52 114 240 494
```

Vnořené cykly

- řídicí struktury můžeme zanořovat, např.:
 - podmínka uvnitř cyklu
 - cyklus uvnitř cyklu

Počet zanoření je neomezený, ale...

```
1  n = 10
2  total = 0
4  for i in range(1, n+1):
5      for j in range(n):
6          print(i+j, end=" ")
7          total += i+j
8      print()
10 print("The result is", total)
```

lec02/table.py

Příklad – hezčí formátování

```
1 n = 8
3 for i in range(1, n+1):
4     for j in range(n):
5         print(str(i+j).ljust(2), end=" ")
6     print()
```

lec02/table2.py

```
1 2 3 4 5 6 7 8
2 3 4 5 6 7 8 9
3 4 5 6 7 8 9 10
4 5 6 7 8 9 10 11
5 6 7 8 9 10 11 12
6 7 8 9 10 11 12 13
7 8 9 10 11 12 13 14
8 9 10 11 12 13 14 15
```

Cyklus while

- Neznámý počet opakování, provádí příkazy dokud platí podmínka

```
while podminka:  
    prikaz
```

- Může se stát:
 - neprovede příkazy ani jednou
 - provádí příkazy do nekonečna (nikdy neskončí)

To většinou znamená chybu v programu

```
3 | n = int(input())  
5 | f = 1  
7 | while n > 0:  
8 |     f = f * n  
9 |     n = n - 1  
11 | print(f)
```

Přerušení cyklu – příkazy `break` a `continue`

- `break` přeruší cyklus

```
1 | for i in range(5):  
2 |     if i==3:  
3 |         break  
4 |     print(i, end=',')
```

0 1 2

- `continue` přeruší aktuální iteraci a začne následující

```
1 | for i in range(5):  
2 |     if i==3:  
3 |         continue  
4 |     print(i, end=',')
```

0 1 2 4

Příklad – binární zápis čísla

```
1  n = int(input())
3  output = ""
5  while n > 0:
6      if n % 2 == 0:
7          output = "0" + output
8      else:
9          output = "1" + output
10     n = n // 2 # integer division
12 print(output)
```

lec02/dec2bin.py

Část II

Funkce

Strukturovaný kód

Programy nepíšeme jako jeden dlouhý štrůdl, ale snažíme se je strukturovat

Proč?

- opakované provádění stejného (velmi podobného) kódu na různých místech algoritmu
- modularita (viz Lego kostky), znovupoužitelnost
- snažší uvažování o problému, čitelnost, dělba práce

Prostředky pro strukturování kódu

- Bloky kódu – (*oddělené odsazením*), např:

```
1 | for i in range(10):  
2 |     print("Budu se pilne ucit.")
```

- Funkce
- Moduly, programy (soubory)

Příklad existujících funkcí – knihovna `math`

- použití knihovny: `import math`
- zaokrouhlování: `round`, `math.ceil`, `math.floor`
- absolutní hodnota: `abs`
- `math.exp`, `math.log`, `math.sqrt`
- goniometrické funkce: `math.sin`, `math.cos`, ...
- konstanty: `math.pi`, `math.e`

```
>>> import math
>>> math.sqrt(4.0)
2.0
>>> math.sin(30./180.*math.pi)
0.49999999999999994
>>> math.exp(1.0)
2.718281828459045
```

Definice uživatelských funkcí

- Příklad – druhá mocnina

```
1 | def square(x):  
2 |     return x*x
```

```
>>> square(3)  
9
```

- Obecně:

```
1 | def jmeno ( parametry ):  
2 |     <blok kodu>
```

- `return(value)` – návrat do nadřazené funkce
- problémy bychom měli členit na podroblémy – zde např. můžeme funkci využít pro výpočet přepony

Příklad uživatelské funkce – binární zápis čísla

```
1 def to_binary(n):
2     output = ""
3
4     while n > 0:
5         if n % 2 == 0:
6             output = "0" + output
7         else:
8             output = "1" + output
9         n = n // 2
10
11     return output
13 print(to_binary(22))
```

lec02/dec2bin-func.py

Vlastnosti funkcí

- **vstup:** parametry funkce
- **výstup:** návratová hodnota (předaná zpět pomocí `return`)
 - `return` není `print`
 - upozornění: `yield` – podobné jako `return`, pokročilý konstrukt, v tomto kurzu nebudeme nepoužívat
- proměnné v rámci funkce:
 - lokální: dosažitelné pouze v rámci funkce
 - globální: dosažitelné všude, minimalizovat použití (více později)
- funkce mohou volat další funkce
 - po dokončení vnořené funkce se interpret vrací a pokračuje
- **rekurze:** volání sebe sama, cyklické volání funkcí (podrobněji později)

Příklad – vnořené volání funkcí

```
1 def parity_info(number):
2     print("Number", number, end=" ")
3     if number % 2 == 0:
4         print("is even")
5     else:
6         print("is odd")
7
8 def parity_experiment(a, b):
9     print("The first number", a)
10    parity_info(a)
11    print("The second number", b)
12    parity_info(b)
13    print("End")
14
15 parity_experiment(3, 18)
```

Funkce – speciality Pythonu

```
1 | def test(x, y=3):  
2 |     print("X =", x, ", Y =", y)
```

- defaultní hodnoty parametrů
- volání pomocí jmen parametrů
- funkci test lze volat např.

```
test(2, 8)
```

```
X = 2, Y = 8
```

```
test(1)
```

```
X = 1, Y = 3
```

```
test(y=5, x=4)
```

```
X = 4, Y = 5
```

- (dále též libovolný počet parametrů a další speciality)

Návrh funkcí

- Specifikace: vstupně-výstupní chování
 - ujasnit si před psaním samotného kódu
 - jaké potřebuje funkce vstupy?
 - co je výstupem funkce
- Funkce by měly být krátké:
 - jedna myšlenka
 - max na jednu obrazovku
 - jen pár úrovní zanoření
- Příliš dlouhá funkce – rozdělit na kratší
- Funkce **vrací hodnotu** vypočítanou ze **vstupních argumentů**.
- **Čistá funkce** (pure function) – výstup závisí pouze na vstupních parametrech, a to jednoznačně.

Příklad – tisk šachovnice

```
#.#.#.#.  
.#.#.#.#  
#.#.#.#.  
.#.#.#.#  
#.#.#.#.  
.#.#.#.#  
#.#.#.#.  
.#.#.#.#
```

Tisk šachovnice – první řešení

```
1 def chessboard(n):
2     for i in range(n):
3         if (i % 2 == 0): even_line(n)
4         else: odd_line(n)
5
6 def even_line(n):
7     for j in range(n):
8         if (j % 2 == 0): print("#", end="")
9         else: print(".", end="")
10    print()
11
12 def odd_line(n):
13    for j in range(n):
14        if (j % 2 == 1): print("#", end="")
15        else: print(".", end="")
16    print()
17
18 chessboard(8)
```

Tisk šachovnice – lepší řešení

```
1 def chessboard(n):
2     for i in range(n):
3         line(n, i % 2)
5 def line(n, parity):
6     for j in range(n):
7         if (j % 2 == parity): print("#", end="")
8         else: print(".", end="")
9     print()
```

lec02/chessboard2.py

```
1 def chessboard(n):
2     for i in range(n):
3         for j in range(n):
4             c = "#" if ((i+j) % 2 == 0) else "."
5             print(c, end="")
6     print()
```

lec02/chessboard3.py_{9/48}

Počet parametrů

```
1 def funkce(a, b, c):  
2     <blok kodu>
```

- Počet parametrů je dán hlavičkou funkce
- Existují i funkce bez parametrů
- Parametry mohou mít defaultní hodnotu

Nemůže být čistá, pokud není konstatní.

Existují i funkce s proměnným počtem parametrů.

Návratová hodnota

Příkazů `return` může být ve funkci několik

```
1  def isprime(n):
2      p=2
3      while p*p<=n:
4          if n % p == 0:
5              return False
6          p+=1
7      return True
```

```
>>> isprime(16)
False
>>> isprime(17)
True
```

Funkce nemusí vracet nic, pak často hovoříme o proceduře nebo void funkci. V proceduře můžeme explicitně specifikovat **return None**

Více návratových hodnot

```
1 | def f(x):  
2 |     return x,2*x,3*x
```

```
>>> a, b, c = f(10)
```

```
>>> a
```

```
10
```

```
>>> b
```

```
20
```

```
>>> c
```

```
30
```

`(x,2*x,3*x)` je objekt typu `tuple` (uspořádaná n-tice)

Rozsah platnosti proměnných

Proměnné definované

- v hlavním programu (mimo funkce) jsou **globální**, viditelné všude
- uvnitř funkce jsou **lokální**, viditelné pouze uvnitř této funkce
- lokální proměnná se stejným jménem **zastíní** proměnnou globální

```
1  b=3 # je globalni
2  def f(x):
3      a=2*x # a je lokalni
4  print(a,b) # otestujme zmenu odsazeni
5
```

```
>>> f(1)
```

```
2 3
```

```
>>> print(b)
```

```
3
```

```
>>> print(a)
```

```
NameError: name 'a' is not defined
```

Funkce jako argument

1. příklad

```
1 def twice(f,x):
2     return f(f(x))
3
4 def square(x):
5     return x*x
6
7 print(twice(square,10))
```

2. příklad

```
1 def repeatNtimes(f,n):
2     for i in range(n): f()
3
4 def ahoj():
5     print("Ahoj")
6
```

Část III

Ladění

Poznámka o ladění

- laděním se nebudeme (na přednáškách) příliš zabývat
- to ale neznamená, že není důležité ...

Ladění je dvakrát tak náročné, jak psaní vlastního kódu. Takže pokud napíšete program tak chytře, jak jen umíte, nebudete schopni jej odladit. (Brian W. Kernighan)

Jak na to?

- ladící výpisy
 - např. v každé iteraci cyklu vypisujeme stav proměnných
 - doporučeno vyzkoušet na ukázkových programech z přednášek
- použití debuggeru
 - dostupný přímo v IDLE
 - sledování hodnot proměnných, spuštěných příkazů, breakpointy, ...
- kompozice na funkce
 - chyba se daleko lépe hledá v dílčí funkci než v celém programu najednou

Čtení chybových hlášek

```
Traceback (most recent call last):
File "sorting.py", line 63, in <module>
test_sorts()
File "sorting.py", line 59, in test_sorts
sort(a)
File "sorting.py", line 52, in insert_sort
a[j] = curent
NameError: name 'curent' is not defined
```

- kde je problém? (identifikace funkce, číslo řádku)
- co je za problém (typ chyby)

Základní typy chyb

- `SyntaxError`
- invalid syntax: zapomenutá dvojtečka či závorka, záměna `=` a `==`, ...
 - EOL while scanning string literal: zapomenutá uvozovka

`NameError` – špatné jméno proměnné (překlep v názvu, chybějící inicializace)

`IndentationError` – špatné odsazení

`TypeError` – nepovolená operace (sčítání čísla a řetězce, ...)

`IndexError` – chyba při indexování řetězce, seznamu, ...

Časté chyby

- zapomenuté formátovací znaky – dvojtečka, apostrof, ...
- špatný počet argumentů funkce
- `"True"` místo `True`
- záměna `print` a `return`