

Programování: základní konstrukce

IB111 Základy programování

Radek Pelánek

2018

Rozcvička

1	2	3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13	14	15
7	8	9	10	11	12	13	14	15	16
8	9	10	11	12	13	14	15	16	17
9	10	11	12	13	14	15	16	17	18
10	11	12	13	14	15	16	17	18	19

Základní konstrukce

- proměnné, operace
- řízení toku výpočtu:
 - podmínovací příkaz (if-else)
 - cykly (for, while)
- funkce

O přednášce

- důraz na princip použití (k čemu to je), ilustrace použití, přemýšlení o problému
- **ilustrace na příkladech**
- zdůraznění častých problémů
- syntax (zápis) jen zběžně, zdaleka ne vše z jazyka Python
- syntax je však potřeba také umět!
 - cvičení
 - samostudium, např. `howto.py.cz`
- některé aspekty zjednodušeně, později v kurzu podrobněji

Úvodní poznámky k syntaxi

- „case sensitive“ – velikost písmen důležitá
- diakritika, české názvy – Python 3 umožňuje, ale v tomto předmětu se držíme anglických písmen a názvů
- komentáře – symbol #

Proměnné: příklad

Problém: vypočítat výšku mostu na základě času pádu koule

vstup: čas

výstup: výška

```
t = float(input())  
h = 0.5 * 10 * t * t  
print(h)
```

Proměnné

- udržují hodnotu
- udržovaná hodnota se může měnit – proto *proměnné*
- typy:
 - číselné: int, float, ...
 - pravdivostní hodnota (bool)
 - řetězec (string)
 - seznam / pole
 - slovník
 - ...

Názvy proměnných

- posloupnost písmen, číslic a znaků '_'
- bez mezer, více slov pomocí:
 - `lower_case_with_underscores`, `CamelCase`,
`mixedCase`
- nelze použít klíčová slova (`if`, `while`, `and`, `True`, ...)
- lze použít některé další prvky jazyka (např. názvy typů), ale není to vůbec dobrý nápad

Výrazy a operace

- výrazy: kombinace proměnných a konstant pomocí operátorů
- operace:
 - aritmetické: sčítání, násobení, ...
 - logické: and, or, not, ...
 - zřetězení řetězců
 - ...
- preference operátorů, pořadí vyhodnocování

Proměnné a výrazy: příklady

```
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
```

```
z = x + s      # error: cannot add int and string
```

Zápis v Pythonu: přiřazení, rovnost

- přiřazení =
x = 3 znamená „přiřad' do x hodnotu 3“
- test na rovnost ==
x == 3 znamená „otestuj zda v x je hodnota 3“
- **častá chyba**: záměna = a ==
- varování: is
x is 3 se chová zdánlivě jako ==, ale jsou tam rozdíly
pokročilý programovací prvek, teď **nepoužívat!**

Zápis v Pythonu: operace

- většina operací „intuitivní“: +, -, *, /, and, or
- umocňování: **
- dělení se zbytkem: %
- celočíselné dělení: //
- zkrácený zápis: „y += 5“ znamená „y = y + 5“

Pořadí vyhodnocování

`3 + 2**3 < 15 or "pes" == "kos"`

- pořadí vyhodnocování vesměs intuitivní („jako v matematice“)
- pokud na pochybách:
 - konzultujte dokumentaci
 - závorkujte
- zkrácené vyhodnocování: `1+1 == 2 or x == 3`

Operace, pořadí vyhodnocování

Operator	Description
<code>lambda</code>	Lambda expression
<code>if - else</code>	Conditional expression
<code>or</code>	Boolean OR
<code>and</code>	Boolean AND
<code>not X</code>	Boolean NOT
<code>in, not in, is, is not, <, <=, >, >=, <>, !=, ==</code>	Comparisons, including membership tests and identity tests,
<code> </code>	Bitwise OR
<code>^</code>	Bitwise XOR
<code>&</code>	Bitwise AND
<code><<, >></code>	Shifts
<code>+, -</code>	Addition and subtraction
<code>*, /, //, %</code>	Multiplication, division, remainder [8]
<code>+x, -x, ~x</code>	Positive, negative, bitwise NOT
<code>**</code>	Exponentiation [9]
<code>x[index], x[index:index], x(arguments...), x.attribute</code>	Subscription, slicing, call, attribute reference
<code>(expressions...), [expressions...], {key:datum...}, `expressions...`</code>	Binding or tuple display, list display, dictionary display, string conversion

Sekvenční řazení příkazů

- řádky pod sebou
- alternativně oddělit středníkem (avšak nedoporučeno)
 $x = 2; y = 3$
- varování: and neslouží k sekvenčnímu řazení
NE: $x = 2 \text{ and } y = 3$

Typové systémy programovacích jazyků

Označuje typy programátor v kódu?

- **explicitní** – programátor v kódu deklaruje typ
např. `int x;`
- **implicitní** – interpret určuje typ automaticky

Jak se provádí typová kontrola?

- **statické** – na základě kódu (při kompilaci, před interpretací)
- **dynamické** – za běhu programu

Typy v Pythonu

- dynamické implicitní typování
 - typ se určuje automaticky
 - typ proměnné se může měnit
 - rozdíl oproti statickému explicitnímu typování (většina kompilovaných jazyků jako C, Pascal, Java)
- „deklarace“ proměnné: první přiřazení hodnoty
- zjištění typu: funkce `type`, `isinstance`
- možnost explicitního přetypování
 - `x = float(3); s = str(158)`

Proměnné a paměť

x 1

$x = x + 1$

x 2

x \longrightarrow 1

$x = x + 1$

x \searrow 1
2

(více později)

- základní výpis: `print(x)`
- více hodnot: `print(x, y, z, sep=";")`
- bez odřádkování: `print(x, end="")`
- další možnosti: `sys.write`, `.format`, `.rjust` ...

(rozdíl oproti Python 2)

Podmínky: příklad

Příklad: počítání vstupného

vstup: věk

výstup: cena vstupenky

```
if age < 18:  
    price = 50  
else:  
    price = 100
```

Lze zapsat bez použití if? Jak?

Podmíněný příkaz

```
if <podmínka>:  
    příkaz1  
else:  
    příkaz2
```

- podle toho, zda platí podmínka, se provede jedna z větví
- podmínka – typicky výraz nad proměnnými
- else větev nepovinná
- vícenásobné větvení: if - elif - ... - else
(switch v jiných jazycích)

Blok kódu

- co když chci provést v podmíněné větvi více příkazů?
- blok kódu
 - Python: vyznačeno odsazením, standardně **4 znaky**
 - jiné jazyky: složené závorky { }, begin-end

Podmíněný příkaz: příklad

```
if age < 6:  
    print("Free entry")  
elif age < 15:  
    price += 20  
    print("Child")  
elif age < 60:  
    price += 100  
    print("Adult")  
else:  
    price += 50  
    print("Senior")
```

If/elif/else: poznámka

mnohonásobný if/elif/elif/.../elif/else:

- většinou indikace nevhodného zápisu
- elegantnější řešení: cyklus + seznam, slovník...

Cykly: příklady

- vstupné za celou rodinu
- výpis posloupnosti čísel
- výpočet faktoriálu
- převod čísla na binární zápis

- opakované provádění sekvence příkazů
- známý počet opakování cyklu:
 - příkaz `for`
 - „opakuj k krát“
- neznámý počet opakování cyklu:
 - příkaz `while`
 - „opakuj dokud není splněna podmínka“

For, range

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

- 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

Faktoriál pomocí for cyklu

- Co to je faktoriál? K čemu se používá?
- Kolik je „5!“?
- Jak vypočítat „n!“ (n je vstup od uživatele)?

Faktoriál pomocí for cyklu

```
n = int(input())  
f = 1  
  
for i in range(1, n+1):  
    f = f * i  
  
print(f)
```

První posloupnost z úvodní přednášky

1 0 0 2 8 22 52 114 240 494

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

Zanořování

- řídicí struktury můžeme zanořovat, např.:
 - podmínka uvnitř cyklu
 - cyklus uvnitř cyklu
 - ...
- libovolný počet zanoření (ale ...)

Rozcvička

1	2	3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13	14	15
7	8	9	10	11	12	13	14	15	16
8	9	10	11	12	13	14	15	16	17
9	10	11	12	13	14	15	16	17	18
10	11	12	13	14	15	16	17	18	19

Rozcvička programem

```
n = 10
total = 0

for i in range(1, n+1):
    for j in range(n):
        print(i+j, end=" ")
        total += i+j
    print()

print("The result is", total)
```

Rozcvička programem – hezčí formátování

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

While cyklus

```
while <podmínka>:  
    příkazy
```

- provádí příkazy dokud platí podmínka
- může se stát:
 - neprovede příkazy ani jednou
 - provádí příkazy do nekonečna (nikdy neskončí) – to většinou znamená chybu v programu

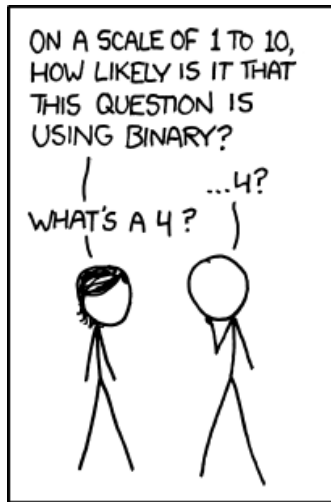
Faktoriál pomocí while cyklu

```
n = int(input())
f = 1
while n > 0:
    f = f * n
    n = n - 1

print(f)
```

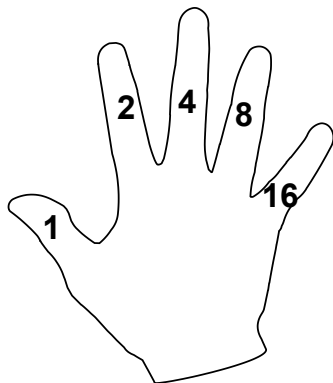
Další příkazy pro řízení cyklů

- `break` – ukončí provádění cyklu (nejvnitřnějšího)
 - `continue` – ukončí aktuální iteraci cyklu a přejde na další
 - `else` větev může být i u cyklu
-
- někdy užitečné, ale snadno zneužitelné (nečitelný kód)
 - v rámci IB111 nepoužíváme



<https://xkcd.com/953/>

Binární soustava: počítání na prstech



<https://www.khanacademy.org/math/vi-hart/v/binary-hand-dance>

Příklad: převod na binární zápis

Problém: převodník z desítkové na binární soustavu

vstup: číslo v desítkové soustavě

výstup: číslo v binární soustavě

- Jak převedeme „22“ na binární zápis?
- Jak převedeme obecné číslo na binární zápis?

Převod na binární zápis

```
n = int(input())
output = ""
while n > 0:
    if n % 2 == 0:
        output = "0" + output
    else:
        output = "1" + output
    n = n // 2  # integer division
print(output)
```

Převod na binární zápis – průběh výpočtu

n = 22	output =
n = 11	output = 0
n = 5	output = 10
n = 2	output = 110
n = 1	output = 0110
n = 0	output = 10110

Funkce

Programy nepíšeme jako jeden dlouhý „štrúdl“, ale dělíme je do funkcí.

Proč?

Programy nepíšeme jako jeden dlouhý „štrúdl“, ale dělíme je do funkcí.

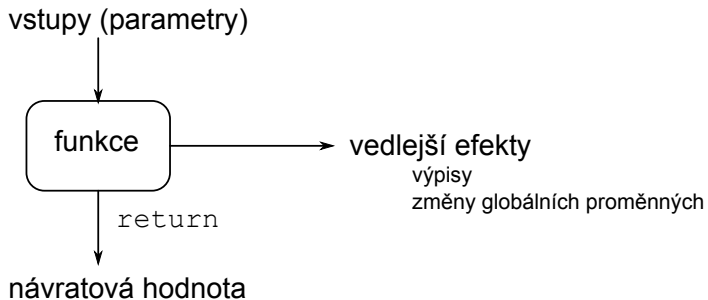
Proč?

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

Funkce pro převod na binární zápis

```
def to_binary(n):  
    output = ""  
    while n > 0:  
        if n % 2 == 0:  
            output = "0" + output  
        else:  
            output = "1" + output  
        n = n // 2  
    return output  
  
print(to_binary(22))
```

Funkce



Funkce

- vstup: parametry funkce
- výstup: návratová hodnota (předána zpět pomocí `return`)
 - `return` není `print`
 - upozornění: `yield` – podobné jako `return`, pokročilý konstrukt, v tomto kurzu nepoužívat
- proměnné v rámci funkce:
 - lokální: dosažitelné pouze v rámci funkce
 - globální: dosažitelné všude, minimalizovat použití (více později)

Vnořené volání funkcí

- funkce mohou volat další funkce
- po dokončení vnořené funkce se interpret vrací a pokračuje
- rekurze: volání sebe sama, cyklické volání funkcí (podrobněji později)

Vnořené volání: jednoduchý příklad

```
def parity_info(number):  
    print("Number", number, end=" ")  
    if number % 2 == 0:  
        print("is even")  
    else:  
        print("is odd")
```

```
def parity_experiment(a, b):  
    print("The first number", a)  
    parity_info(a)  
    print("The second number", b)  
    parity_info(b)  
    print("End")
```

```
parity_experiment(3, 18)
```

Vnořené volání – ilustrace

```
parity_experiment(3, 18)
```

```
def parity_experiment(a, b):  
    print("The first number", a)  
    parity_info(a)  
    print("The second number", b)  
    parity_info(b)  
    print("End")
```

```
def parity_info(number):  
    print("Number", number, end=" ")  
    if number % 2 == 0:  
        print("is even")  
    else:  
        print("is odd")
```

Rozcvička programem II

Experimentální otestování hypotézy o třetí mocnině

```
def table_sum(n):
    total = 0
    for i in range(1, n+1):
        for j in range(n):
            total += i+j
    return total

def check_sums(how_many):
    for n in range(1, how_many+1):
        print(n, n**3, table_sum(n), sep="\t")
```

Druhá posloupnost z úvodní přednášky

0 1 1 2 1 2 2 3 1 2 2 3 2 3 3 4 1 2 2 3 2 3 3 4 2
3 3 4 3 4 ...

Druhá posloupnost z úvodní přednášky

0 1 1 2 1 2 2 3 1 2 2 3 2 3 3 4 1 2 2 3 2 3 3 4 2
3 3 4 3 4 ...

Počet jedniček v bitovém zápisu.

Jak vypsát programem?

Druhá posloupnost z úvodní přednášky

```
def count_ones(n):  
    count = 0  
    while n > 0:  
        if n % 2 == 1: count += 1  
        n = n // 2  
    return count  
  
def sequence(n):  
    for i in range(n):  
        print(count_ones(i), end=" ")
```

Funkce: Python speciality

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

- defaultní hodnoty parametrů
- volání pomocí jmen parametrů
- test můžeme volat např.:
 - test(2, 8)
 - test(1)
 - test(y=5, x=4)
- (dále též libovolný počet parametrů a další speciality)

Funkce: rady

- specifikace: vstupně-výstupní chování – ujasnit si před psáním 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 menší

Programátorská kultura

- psát **smysluplné komentáře**, dokumentační řetězce (viz později)
- dávat proměnným a funkcím **smysluplná jména**
- neopakovat se, nepoužívat „copy&paste kód“
- doporučení k úpravě kódu v Pythonu – „PEP8“:
<https://www.python.org/dev/peps/pep-0008/>

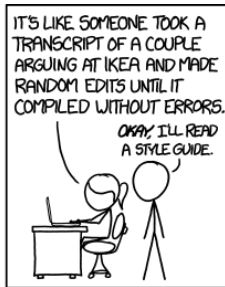
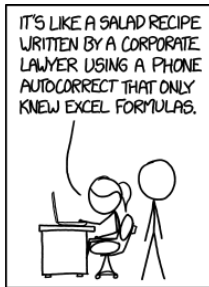
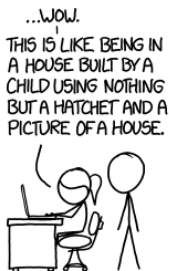
PEP = Python Enhancement Proposals

průběžné téma kurzu

Proměnné, funkce, komentáře: jazyk

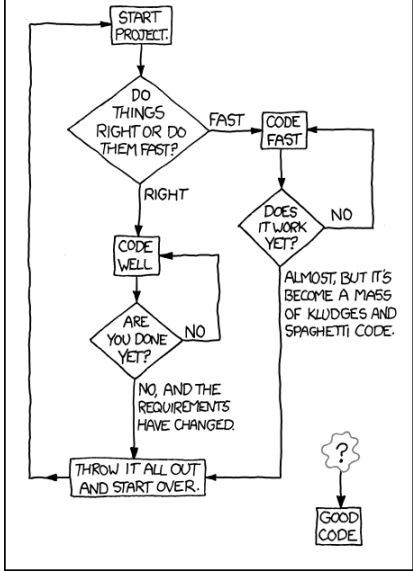
- doporučení (užitečný návyk hned od začátku): **vše anglicky**
- české názvy: míchání jazyků (klíčová slova anglicky), nekonzistence s diakritikou, ...
- větší projekty – angličtina nutnost
- *style guide: Python coders from non-English speaking countries: please write your comments in English, unless you are 120% sure that the code will never be read by people who don't speak your language.*

... ale nezasekněte se na tom



<https://xkcd.com/1513/>
viz též <https://xkcd.com/1695/>, <https://xkcd.com/1833/>

HOW TO WRITE GOOD CODE:



Další důležité programátorské konstrukce

- složitější datové typy (seznamy, řetězce), objekty
- vstup/výstup (input/output, IO):
 - standardní IO
 - soubory
- dělení projektu do více souborů (packages), použití knihoven

viz další přednášky, cvičení, samostudium

Příklad: výpis šachovnice

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

Řešení funkční, ne úplně vhodné

```
def chessboard(n):  
    for i in range(n):  
        if (i % 2 == 0): even_line(n)  
        else: odd_line(n)  
  
def even_line(n):  
    for j in range(n):  
        if (j % 2 == 0): print("#", end="")  
        else: print(".", end="")  
    print()  
  
def odd_line(n):  
    for j in range(n):  
        if (j % 2 == 1): print("#", end="")  
        else: print(".", end="")  
    print()
```

Lepší řešení

```
def chessboard(n):  
    for i in range(n):  
        line(n, i % 2)  
  
def line(n, parity):  
    for j in range(n):  
        if (j % 2 == parity): print("#", end="")  
        else: print(".", end="")  
    print()
```


Jiný zápis

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

DRY vs WET programming

- DRY = Don't Repeat Yourself
- WET
 - Write Everything Twice
 - We Enjoy Typing
 - Waste Everyone's Time

Příklad: Hádanka hlavy a nohy

Farmář chová prasata a slepice. Celkem je na dvoře 20 hlav a 56 noh. Kolik má slepic a kolik prasat?

- jak vyřešit pro konkrétní zadání?
- jak vyřešit pro obecné zadání (H hlav a N noh)?
- co když farmář chová ještě osminohé pavouky?

Hlavy, nohy: řešení

dva možné přístupy:

- 1 „inteligentně“: řešení systému lineárních rovnic
- 2 „hrubou silou“:
 - „vyzkoušej všechny možnosti“
 - cyklus přes všechny možné počty prasat

Hlavy, nohy: program

```
def solve(heads, legs):  
    for pigs in range(0, heads+1):  
        hens = heads - pigs  
        if pigs * 4 + hens * 2 == legs:  
            print ("pigs =", pigs,  
                  "hens =", hens)
```

```
solve(20, 56)
```

Jak bych musel program změnit, kdybych řešil úlohu i s pavouky?

Shrnutí

- základní konstrukce
 - proměnné
 - řízení toku výpočtu: if-else, for, while
 - funkce
- programátorská kultura

příště: počítání s čísly