FAKULTA
INFORMATIKY
Masarykova univerzita

IB111 Základy programovania

Cvičenie 2 – výrazy, výstup, riadiace príkazy, procedúry

Matej Troják

(Poďakovanie: Valdemar Švábenský)

1. Premenné a dátove typy
2. Výstup na obrazovku
3. Podmienené príkazy
4. Príkazy cyklu
5. Funkcie (a.k.a. procedúry)
6. Konvencie písania kódu
7. Príklady: textová grafika
8. Príklady: výpisy postupností
9. Príklady: tabuľky

Premenná

- meno pre hodnotu uloženú v pamäti
- hodnota sa môže meniť (preto názov “premenná”)

```
>>> age = 1
>>> age
1
>>> age = 900
>>> age
900
```

- k dispozícii znaky a/A–z/Z, 0–9 a _, nie `keywords`
- premenná pomáha čitateľnosti kódu (pre človeka)

Dátový typ

- premenné môžu byť rôzneho typu (číslo, text, . . .)
- typ hodnoty premennej sa môže meniť (dynamické typovanie)
- operácie, ktoré sa s nimi dajú spraviť

```
>>> age = 'Some text.'
>>> age
'Some text.'
```

Príklady:

- Celé číslo (`int`): 42, 23, 0, -14
- Reálne číslo (`float`): 42.0, 0.21, -7.59876, 3e8
- Logická hodnota (`bool`): True, False
- Reťazec (`str`): "The answer is 42!", '0.21'
- Zoznam (`list`): [1, 4, 9, 16, 25]

  . . .

Dátový typ

- Zistenie typu: funkcia `type()`

```
>>> type(1)
<class 'int'>
>>> type('Hello')
<class 'str'>
```

- Pretypovanie: funkcie `int()`, `float()`, `str()`, ...

```
>>> int(3.14159)
3
>>> float(5)
5.0
>>> str(1)
'1'
```

Dátový typ

- pozor na operácie s nekompatibilnými typmi

```
>>> a = 1
>>> b = '2'
>>> a + b
TypeError: unsupported operand type(s)
for +: 'int' and 'str'
>>> type(a)
<class 'int'>
>>> type(b)
<class 'str'>
```

Dátový typ

- Riešenie: pretypovanie:

```
>>> a + int(b)
3
```

- pozor na nemožné pretypovanie

```
>>> a = 'hello'
>>> float(a)
ValueError: could not convert string to float: 'hello'
```

Operátor

- Jazyková konštrukcia umožňujúca operácie na hodnotách
- Jeho argument: operand

Aritmetické operátory

- Úloha: aké hodnoty budú v premenných a až j?

```
>>> a = 5 + 2                    # 5+2
>>> b = 5 - 2                    # 5-2
>>> c = 5 * 2                    # 5*2
>>> d = 5 / 2                    # 5/2
>>> e = int(5 / 2)               # int(5/2)
>>> f = 5 // 2                   # 5//2
>>> g = int(-5 / 2)              # int(-5/2)
>>> h = -5 // 2                  # -5//2
>>> i = 5 % 2                    # 5%2
>>> j = 5 ** 2                   # 5**2
```

- Často je dobré kvôli prehľadnosti oddeliť operátor medzerou z oboch strán (zvážte čitateľnosť)

Aritmetické operátory

- Úloha: aké hodnoty budú v premenných a až j?

```
>>> a = 5 + 2                    # 7
>>> b = 5 - 2                    # 3
>>> c = 5 * 2                    # 10
>>> d = 5 / 2                    # 2.5
>>> e = int(5 / 2)               # 2
>>> f = 5 // 2                   # 2
>>> g = int(-5 / 2)              # -2
>>> h = -5 // 2                  # -3
>>> i = 5 % 2                    # 1
>>> j = 5 ** 2                   # 25
```

Skrátené zápisy

- Skrátené operátory priradenia: je rozdiel medzi týmito blokmi kódu?

```
>>> a = 2
>>> a = a + 5
>>> a
7
```

```
>>> a = 2
>>> a += 5
>>> a
7
```

- Operátory priradenia na jednom riadku

```
>>> a, b = 2, 1
>>> a
2
>>> b
1
```

Skrátené zápisy

- Úloha: je rozdiel medzi týmito blokmi kódu?

```
>>> a, b = 2, 1
>>> a = b
>>> b = a + b
>>> print(a)
# ???
>>> print(b)
# ???
```

```
>>> a, b = 2, 1
>>> a, b = b, a + b
# Assign on one line
>>> print(a)
# ???
>>> print(b)
# ???
```

Skrátené zápisy

- Úloha: je rozdiel medzi týmito blokmi kódu?

```
>>> a, b = 2, 1
>>> a = b
>>> b = a + b
>>> print(a)
1
>>> print(b)
2
```

```
>>> a, b = 2, 1
>>> a, b = b, a + b
# Assign on one line
>>> print(a)
1
>>> print(b)
3
```

Relačné operátory

- Pre porovnávanie hodnôt
- Úloha: aké hodnoty budú v premenných a až g?

```
>>> a = 4 == 4
>>> b = 8 != 5
>>> c = 9 < 4
>>> d = 9 <= 4
>>> e = 5 > 3
>>> f = 5 >= 5
>>> g = 1 < 2 < 3
```

Relačné operátory

- Pre porovnávanie hodnôt
- Úloha: aké hodnoty budú v premenných a až g?

```
>>> a = 4 == 4
>>> b = 8 != 5
>>> c = 9 < 4
>>> d = 9 <= 4
>>> e = 5 > 3
>>> f = 5 >= 5
>>> g = 1 < 2 < 3
```

- Riešenie je typu `bool`

Relačné operátory

- Pre porovnávanie hodnôt
- Úloha: aké hodnoty budú v premenných a až g?

```
>>> a = 4 == 4          # True
>>> b = 8 != 5          # True
>>> c = 9 < 4           # False
>>> d = 9 <= 4          # False
>>> e = 5 > 3           # True
>>> f = 5 >= 5          # True
>>> g = 1 < 2 < 3       # True
```

- Riešenie je typu `bool`

Logické operátory

- Povinnosť oddeliť operátor medzerou z oboch strán
- Úloha: aké hodnoty budú v premenných a až i?

```
>>> a = True and True
>>> b = True and False
>>> c = False and False
>>> d = True or True
>>> e = True or False
>>> f = False or False
>>> g = not True
>>> h = not False
>>> i = not not True
```

Logické operátory

- Povinnosť oddeliť operátor medzerou z oboch strán
- Úloha: aké hodnoty budú v premenných a až i?

```
>>> a = True and True
>>> b = True and False
>>> c = False and False
>>> d = True or True
>>> e = True or False
>>> f = False or False
>>> g = not True
>>> h = not False
>>> i = not not True
```

- Riešenie je typu `bool`

Logické operátory

- Povinnosť oddeliť operátor medzerou z oboch strán
- Úloha: aké hodnoty budú v premenných a až i?

```
>>> a = True and True          # True
>>> b = True and False         # False
>>> c = False and False        # False
>>> d = True or True           # True
>>> e = True or False          # True
>>> f = False or False         # False
>>> g = not True               # False
>>> h = not False              # True
>>> i = not not True           # True
```

- Riešenie je typu `bool`

Tvorba výrazov

- Kombinácia konštánt, premenných, operátorov a funkcií

```
>>> 5 + 4**2 >= 5 + 4*2
```

- dôležitá je priorita operátorov: aritmetické < relačné < logické

```
>>> not 5 + 16 >= 5 + 4*2
False
>>> not ((5 + 16) >= (5 + (4*2)))
False
```

1. Premenné a dátove typy
2. Výstup na obrazovku
3. Podmienené príkazy
4. Príkazy cyklu
5. Funkcie (a.k.a. procedúry)
6. Konvencie písania kódu
7. Príklady: textová grafika
8. Príklady: výpisy postupností
9. Príklady: tabuľky

Vypisovanie výstupu

- Funkcia `print()`
  - Vyhodnotí daný výraz, vypíše ho a ukončí riadok (znak `\n`)

```
>>> a = 1
>>> b = 2
>>> print(a + b)
3
>>> a = '1'
>>> b = '2'
>>> print(a + b)
12
```

Vypisovanie výstupu

- Funkcia `print()` – parameter `sep`
  - Nastavuje oddeľovač medzi vypisovanými hodnotami
  - default: `sep=''` (medzera)

```
>>> a = 1
>>> b = 2
>>> print(a, b)
1 2
>>> print(a, b, sep=',')
1,2
```

Vypisovanie výstupu

- Funkcia `print()` – parameter end
  - Nastavuje oddeľovač medzi vypisovanými hodnotami
  - default: `end='\n'` (nový riadok)

```
>>> for i in range(3):
        print('*')
*
*
*
>>> for i in range(3):
        print('*', end=' ')
* * *
```

Vypisovanie výstupu (vhodné pre bonusové úlohy)

- Funkcia `format()`
  - operácia nad reťazcami
  - default: `end=\n` (nový riadok)

```
>>> name = 'John'
>>> age = 25
>>> print('{0} is {1} years old'.format(name, 25))
John is 25 years old
>>> from math import pi
>>> print('First 5 decimal digits of the Pi are {0:.6}'.format(pi))
First 5 decimal digits of the Pi are 3.14159
```

- Ďalšie príklady:
  www.python-course.eu/python3_formatted_output.php

1. Premenné a dátove typy
2. Výstup na obrazovku
3. Podmienené príkazy
4. Príkazy cyklu
5. Funkcie (a.k.a. procedúry)
6. Konvencie písania kódu
7. Príklady: textová grafika
8. Príklady: výpisy postupností
9. Príklady: tabuľky

Podmienené príkazy

Žena pošle programátora do obchodu a vraví:
„Kúp 10 rožkov, ak budú mať vajcia, tak kúp 6.“
Programátor vojde do obchodu a vraví:
„Dobrý deň, máte vajcia?“
Predavačka odpovie, že áno.
„Tak 6 rožkov poprosím.“

Podmienené príkazy

Žena pošle programátora do obchodu a vraví:
„Kúp 10 rožkov, ak budú mať vajcia, tak kúp 6.“
Programátor vojde do obchodu a vraví:
„Dobrý deň, máte vajcia?“
Predavačka odpovie, že áno.
„Tak 6 rožkov poprosím.“

```
if pocet_vajec > 0:
    nakup = 6 rozkov
else:
    nakup = 10 rozkov
```

Príkaz vetvenia if

- Vykonanie bloku príkazov podľa splnenia (`True`) podmienky (logický výraz)

```
if <condition is True>:
    <do something>
```

- Nesplnená podmienka (`False`): zanorené príkazy sú preskočené
  - Pozor! Na `False` sa v Pythone vyhodnotia aj `0, 0.0, '', [], (), None, ...`

```
if pekne_pocasie == True:
    print('Vyuka sa rusi.')
```

Príkaz vetvenia if-else

- Splnená podmienka: vykonajú sa príkazy vo vetve `if`
- Nesplnená podmienka: vykonajú sa príkazy vo vetve `else`

```
if <condition is True>:
    <do something >
else: # condition is False
    <do something different >
```

```
if pocet_studentov > 5:
    print('Budeme sa ucit.')
else:
    print('Ideme na pivo.')
```

Príkaz vetvenia if-elif

- Splnená podmienka: vykonajú sa príkazy vo vetve `if`
- Nesplnená podmienka: vyhodnotí sa podmienka pri `elif`

```
if <condition is True>:
    <do something >
elif <different condition is True>:
    <do something different >
```

```
if pocet_studentov > 10:
    print('Budeme sa ucit v ucebni.')
elif pocet_studentov > 5:
    print('Budeme sa ucit na nadvori.')
```

Príkaz vetvenia if-elif-else

- Kombinácia predošlých
- Funguje ľubovoľne mnoho `elif`

```
if pocet_studentov > 10:
    print('Budeme sa ucit v ucebni.')
elif pocet_studentov > 5:
    print('Budeme sa ucit na nadvori.')
else:
    print('Tak teda dame to pivo...')
```

- Príliš mnoho if-ov značí, že niečo robíte zle

Príkaz vetvenia – čerešnička

- Tradičné vyhodnotenie podmienky

```
max = 0
if x > y:
    max = x
else:
    max = y
```

- Ekvivalentný zápis (špecifický pre Python)

```
max = x if x > y else y
```

- Význam: „Nech max je x ak x je väčšie ako y, inak nech max je y.“

1. Premenné a dátove typy
2. Výstup na obrazovku
3. Podmienené príkazy
4. Príkazy cyklu
5. Funkcie (a.k.a. procedúry)
6. Konvencie písania kódu
7. Príklady: textová grafika
8. Príklady: výpisy postupností
9. Príklady: tabuľky

Príkaz cyklu while

- Opakovanie kódu, kým je splnená logická podmienka

```
while <condition is True>:
    <do something>
```

- Obvykle v cykle aktualizujeme premennú, ktorá nakoniec spôsobí zneplatnenie podmienky a ukončenie cyklu

```
number = 0
while number < 3:        # Condition
    print(number)
    number += 1
```

- Možný je nekonečný cyklus
- Cyklus nemusí prebehnúť ani raz

Príkaz cyklu while – break a continue

- `break` – okamžité ukončenie cyklu

```
number = 0
while number < 3:       # Condition
    print(number)
    number += 1
    if number >= 3:
        break
```

- `continue` – ukončenie jednej aktuálnej iterácie cyklu

```
number = 0
while number < 3:       # Condition
    print(number)
    number += 1
    if number >= 3:
        continue        # Jump to condition
```

Príkaz cyklu for

- Opakovanie bloku príkazov určený počet krát

```
>>> for number in range(3):
        print(number)
0
1
2
```

- Iteračná premenná `number` postupne nadobúda celočíselné hodnoty z rozsahu $[0, 3)$
- Dá sa vždy prepísať pomocou cyklu `while`

Príkaz cyklu for – možnosti

- `range(a)` – rozsah $[0, a)$
- `range(a, b)` – rozsah $[a, b)$
- `range(a, b, c)` – rozsah $[a, b)$ zvyšovaný o c

```
>>> for number in range(0, 6, 2):
        print(number)
0
2
4
>>> for number in range(6, 3, -1):
        print(number)
6
5
4
```

- Poznámka: `range()` generuje nemeniteľnú postupnosť

Zanorené cykly

- cykly je možné neobmedzene zanorovať a kombinovať
- treba myslieť na výpočtovú náročnosť

```
>>> n = 5
>>> for i in range(n):
        for j in range(n):
            print('{0:2}'.format(j + i*n), end=' ')
        print()         # new line
 0  1  2  3  4
 5  6  7  8  9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24
```

1. Premenné a dátove typy
2. Výstup na obrazovku
3. Podmienené príkazy
4. Príkazy cyklu
5. Funkcie (a.k.a. procedúry)
6. Konvencie písania kódu
7. Príklady: textová grafika
8. Príklady: výpisy postupností
9. Príklady: tabuľky

Funkcie – motivácia

- Pomenovaná sekvencia príkazov
- Python sa „naučí nový príkaz“
- Znovupoužitie kódu s rôznymi hodnotami
- Sprehľadnenie kódu

```
def triangle(length):
    for _ in range(3):      # premenna sa v cykle nepouziva (znak _)
        forward(length)
        left(120)
```

- Definuje sa raz, dá sa použiť nekonečne mnohokrát
- Použitie (tzv. volanie funkcie):

```
>>> triangle(250)
```

Funkcie – definícia

```
def my_funct(param1, param2, ...):      # header
    statement_1                         # \
    ...                                 #  - body
    statement_n                         # /
```

- Kľúčové slovo `def`
- Meno funkcie (povolené meno: ako pre premennú)
- Zoznam parametrov (vstupov) v zátvorkách
- telo funkcie odsadené

Príklad funkcie

```
def say_what(text):
    '''Print desired <text> to console.'''
    print(text)
```

- Zmysluplné a výstižné meno

```
>>> say_what('Hi there!')
Hi there!
>>> help(say_what)
Help on function say_what in module __main__:

say_what(text)
    Print desired <text> to console.
```

- (Okomentované chovanie cez docstring)
  https://www.python.org/dev/peps/pep-0257/

Parametre funkcií

- Formálne parametre: mená vstupov funkcie v definícii

```
def power(base, exponent):
    result = base ** exponent
    print('{} to the power of {} is {}.'.format(base, exponent , result))
```

- Skutočné parametre (argumenty): konkrétne hodnoty, ktoré volajúci príkaz posiela funkcii (v danom poradí)
- Formálne parametre sú pri volaní nahradené skutočnými

```
>>> power(3, 4)
# base initializes to 3, exponent to 4
3 to the power of 4 is 81.
```

Rozsah platnosti (scope)

Globálne premenné:

- viditeľné v celom programe
- náchylné na chyby
- OK pre konštanty

```
>>> a = 10
>>> def foo():
>>>     print(a)
>>> foo()
10
>>> print(a)
10
```

Lokálne premenné:

- viditeľné len v rámci bloku
- po skončení bloku „zomrú“
- „zatienia“ tie globálne

```
>>> a = 10
>>> def foo():
>>>     a = 5
>>>     print(a)
>>> foo()
5
>>> print(a)
10
```

Defaultné hodnoty parametrov

```
>>> def divide_numbers(x, y, preci=3):
        print('Result of {0}/{1} is {2:.{3}}'.format(x, y, x/y, preci))
```

```
>>> divide_numbers(4, 7)
Result of 4/7 is 0.571
>>> divide_numbers(4, 7, 5)
Result of 4/7 is 0.57143
>>> divide_numbers(4, 7, preci = 5)
Result of 4/7 is 0.57143
```

Kľúčové slovo pass

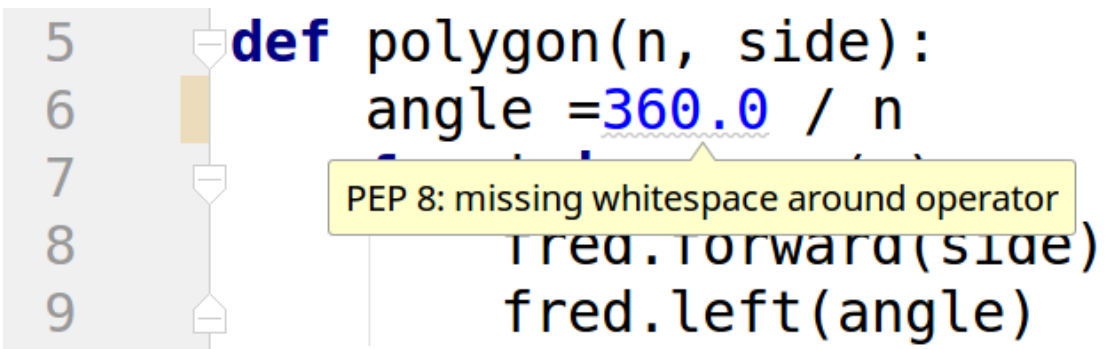
- Prázdna operácia, nič nerobí
- Vypĺňa miesto, kde sa očakáva kód, ale programátor zatiaľ nechce nič písať

```
def foo():
    pass
```

1. Premenné a dátove typy
2. Výstup na obrazovku
3. Podmienené príkazy
4. Príkazy cyklu
5. Funkcie (a.k.a. procedúry)
6. Konvencie písania kódu
7. Príklady: textová grafika
8. Príklady: výpisy postupností
9. Príklady: tabuľky

Konvencie písania kódu

- Konvencie (*code style*) definujú formátovanie kódu
- Zlepšujú čitateľnosť – IT firmách sú vynútené
- V domácich úlohách a písomkách:
  norma PEP8 – `https://www.python.org/dev/peps/pep-0008/`
- Jedná sa len o odporúčania: zdravý rozum víťazí
- Možnosť automatickej kontroly
  napr. pylint (`https://www.pylint.org/`)
- PyCharm veľmi pomáha, počúvajte jeho rady!



Kód píšte vždy v angličtine

- Ako by sa vám páčilo čítať kód po niekom, kto píše fínsky?

```
def toiminto():
    for maara in range(5):
        tulos += maara
    print(tulos)
```

Používajte zdravý rozum ;)

- Ako by sa vám páčilo čítať kód po niekom, kto píše názvy premenných a funkcií takto?

```
def tHisFunCTiOniSBasICalLyUSelEsS():
    for nUMbeR in range(5):
        sUm += nUMbeR
    print(sUm)
```

- versus

```
def this_function_is_basically_useless():
    for number in range(5):
        sum += number
    print(sum)
```

Pravidlá pre funkcie

- Zmysluplná funkcia – http://www.hovnokod.cz/367
- Vhodné meno funkcie – http://www.hovnokod.cz/392
- Vhodné komentáre – http://www.hovnokod.cz/401
- Krátka funkcia – http://www.hovnokod.cz/353

1. Premenné a dátove typy
2. Výstup na obrazovku
3. Podmienené príkazy
4. Príkazy cyklu
5. Funkcie (a.k.a. procedúry)
6. Konvencie písania kódu
7. Príklady: textová grafika
8. Príklady: výpisy postupností
9. Príklady: tabuľky

Úloha 1a

- Napíšte funkciu `square(n)`, ktorá na obrazovku vypíše štvorec o veľkosti $n \times n$ vytvorený zo znakov `#`.

```
>>> square(5)
# # # # #
# # # # #
# # # # #
# # # # #
# # # # #
```

- *Bonus:* Rozšírte funkciu `square(n, char)` o parameter `char`, ktorý údava znak použitý na vykresľovanie (namiesto `#`).

Úloha 1b

- Napíšte funkciu `rectangle(n, m)`, ktorá na obrazovku vypíše obdĺžnik o veľkosti $n \times m$ vytvorený zo znakov `#`.

```
>>> rectangle(5, 7)
# # # # # # #
# # # # # # #
# # # # # # #
# # # # # # #
# # # # # # #
```

Úloha 2

- Napíšte funkciu `empty_square(n)`, ktorá na obrazovku vypíše štvorec o veľkosti $n \times m$ vytvorený zo znakov `#` v strede prázdny.

```
>>> empty_square(5)
# # # # #
#       #
#       #
#       #
# # # # #
```

- *Bonus:* Rozšírte funkciu `empty_square(n, char)` o parameter `char` tak, aby sa namiesto prázdneho priestoru vykresľoval znak `char`.

1. Premenné a dátove typy
2. Výstup na obrazovku
3. Podmienené príkazy
4. Príkazy cyklu
5. Funkcie (a.k.a. procedúry)
6. Konvencie písania kódu
7. Príklady: textová grafika
8. Príklady: výpisy postupností
9. Príklady: tabuľky

Úloha 3

- Napíšte funkciu `first_n(number, n)`, ktorá na obrazovku vypíše prvých n čísel od čísla `number`.

```
>>> first_n(2, 10)
2 3 4 5 6 7 8 9 10 11
>>> first_n(5, 8)
5 6 7 8 9 10 11 12
```

- *Bonus:* Upravte funkciu `first_n(number, n)` tak, aby na obrazovku vypísala prvých n čísel DO čísla `number` (brať do úvahy aj záporné čísla a nulu).

```
>>> first_n(2, 10)
-7 -6 -5 -4 -3 -2 -1 0 1 2
```

Úloha 4

- Napíšte funkciu `powers(n)`, ktorá na obrazovku vypíše prvých `n` mocnín čísla 2.

```
>>> powers(10)
1 2 4 8 16 32 64 128 256 512
```

- *Bonus:* Rozšírte funkciu `powers(base, n)` o parameter `base` tak, aby na obrazovku vypísala prvých `n` mocnín o danom základe `base`.

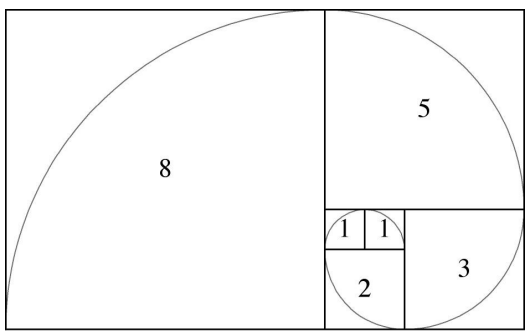
```
>>> powers(3, 8)
1 3 9 27 81 243 729 2187
```

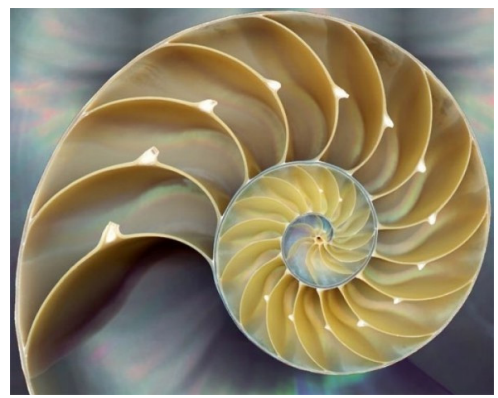
Úloha 5

- Napíšte funkciu `fibonacci(n)`, ktorá na obrazovku vypíše prvých n prvkov Fibonacciho postupnosti, ktorá je definovaná nasledovne:
  - $f(0) = 1$
  - $f(1) = 1$
  - $f(n) = f(n-1) + f(n-2)$

```
>>> fibonacci(10)
1 1 2 3 5 8 13 21 34 55
```

- grafické znázornenie – hrany štvorcov





1. Premenné a dátove typy
2. Výstup na obrazovku
3. Podmienené príkazy
4. Príkazy cyklu
5. Funkcie (a.k.a. procedúry)
6. Konvencie písania kódu
7. Príklady: textová grafika
8. Príklady: výpisy postupností
9. Príklady: tabuľky

Úloha 6

- Napíšte funkciu `table_products(n)`, ktorá vypíše tabuľku s daným počtom riadkov a stĺpcov (+ popisný riadok a stĺpec), kde v každej bunke sa nachádza súčin čísla riadku a stĺpca.

```
>>> table_products(5)
    1 2 3 4 5
    - - - - -
1 | 1 2 3 4 5
2 | 2 4 6 8 10
3 | 3 6 9 12 15
4 | 4 8 12 16 20
5 | 5 10 15 20 25
```

- *Bonus:* Postarajte sa o to, aby bola tabuľka vždy pekne zarovnaná (v prípade, keď sú hodnoty v riadkoch/stĺpcoch viac ciferné.
  Hint: Funkcia `len('string')` spočíta dĺžku zadaného reťazca (6).

Domáca úloha !