

## Python 3 – základní ukázky kódu

[www.python.org](http://www.python.org)

Spustit program: F5 (nebo v menu: Run → Run Module)

### Výpis textu na obrazovku

```
print("Ahoj!")
```

### Uložení hodnoty do proměnné, tisk hodnoty proměnné

```
i = 3
print(i)
print(i*i + 2)
print(10**i)    # umocňování: 10 na i

# zmenšení hodnoty proměnné o 1 a její tisk
print("Hodnota proměnné i = 3 zmenšena o jednu:")
i = i - 1
print(i)

# tisk prázdného řádku
print()
```

### Výpis hodnoty proměnné s komentářem

```
# 1 - primitivní přístup
# na výstupu se vytiskne vše vedle sebe a oddělené mezerami
print("Hodnota je", i, "- to není mnoho.")

# 2 - moderní způsob formátovaného výstupu – nejlepší
print("Hodnota je {} - to není mnoho.".format(i) )

# 3a – tisk spojení řetězců,      spojování řetězců zajišťuje operátor +
# celé číslo i je převedeno na řetězec funkcí str()
print("Hodnota je " + str(i) + " - to není mnoho.")

# 3b – spojení řetězců uloženo do proměnné, ta se pak tiskne
řetězec = "Hodnota je " + str(i) + " - to není mnoho."
print(řetězec)

# 4 - starší způsob formátovaného výstupu
print("Hodnota je %d - to není mnoho." % i)
```

## Cyklus for (pozor na odsazení)

```
# pro i od 0 do 5 (nikoli včetně 5) tiskneme i  
for i in range(5):  
    print(i)
```

Výstup programu:

```
0  
1  
2  
3  
4
```

```
# pro k od 1 do 4  
for k in range(1, 5):  
    print("Číslo:", k)
```

Výstup programu:

```
Číslo: 1  
Číslo: 2  
Číslo: 3  
Číslo: 4
```

```
# pro i od 1 do 5 (nikoli 6) tiskneme i a jeho druhou a třetí mocninu  
for i in range(1, 6):  
    print(i, i*i, i**3)
```

Výstup programu:

```
1 1 1  
2 4 8  
3 9 27  
4 16 64  
5 25 125
```

```
# výpis hodnost oddělených mezerou, ne každá na samostatný řádek  
for i in range(10):  
    print(i, end=' ')
```

Výstup programu:

```
0 1 2 3 4 5 6 7 8 9
```

```
# výpis hodnost oddělených čárkou, ne každá na samostatný řádek  
for i in range(5):  
    print(i, end=', ')
```

```
# výpis hodnost pro i od 2 do 10 (bez 10) s krokem 2
```

```
for i in range(2, 10, 2):  
    print(i, end=' ')
```

Výstup programu:

```
2 4 6 8
```

```
# úloha malého Gausse: součet čísel od 1 do 100
```

```
soucet = 0
```

```
for i in range(1, 101):  
    soucet = soucet + i
```

```
print("součet čísel od 1 do 100: ", soucet)
```

Modifikujte tento program na součet prvních  $n$  členů libovolné aritmetické posloupnosti (se zadanou diferencí  $d$ , prvním členem  $a_1$ ).

Napište program, který vypočte součet prvních  $n$  členů geometrické posloupnosti (se zadaným kvocientem  $q$ , prvním členem  $a_1$ ).

## if – podmínka (pozor na odsazení)

```
číslo = 2
```

```
if číslo > 0:  
    print("Zadané číslo {} je kladné.".format(číslo))  
else:  
    print("Zadané číslo {} není kladné.".format(číslo))
```

Výstup programu:

Zadané číslo 2 je kladné.

```
číslo = -3
```

```
if číslo > 0:  
    print("Zadané číslo {} je kladné.".format(číslo))  
else:  
    print("Zadané číslo {} není kladné.".format(číslo))
```

Výstup programu:

Zadané číslo -3 není kladné.

*# důkladné testování – volba mnoha hodnot ze seznamu*

```
for n in [2, -3, 5, 0, -7, 12]:  
    if n > 0:  
        print("Zadané číslo {} je kladné.".format(n))  
    else:  
        print("Zadané číslo {} není kladné.".format(n))
```

Výstup programu:

Zadané číslo 2 je kladné.  
Zadané číslo -3 není kladné.  
Zadané číslo 5 je kladné.  
Zadané číslo 0 není kladné.  
Zadané číslo -7 není kladné.  
Zadané číslo 12 je kladné.

## if – automatické slovní hodnocení na základě známky

```
známka = int( input("Zadejte známku: ") )
```

```
print("Slovní hodnocení:", end='\t')           # \t - tabulátor, \n - newline (nový řádek)
```

```
if známka == 1:                               # = přiřazovací rovná se,      == porovnávací rovná se  
    print('výborně')  
elif známka == 2:  
    print('chvalitebně')  
elif známka == 3:  
    print('dobře')  
elif známka == 4:  
    print('dostatečně')  
elif známka == 5:  
    print('nedostatečně')  
else:  
    print('takovou známku nemáme')
```

## Definice funkcí

- funkce může (ale nemusí) vracet nějakou hodnotu či více hodnot
- vrácení hodnoty zajišťuje klíčové slovo **return**, po vrácení hodnoty se činnost funkce ukončí
- funkce může (ale nemusí) mít nějaké parametry
- ale vždy musí mít za svým názvem závorky

```
# definujeme si vlastní funkci  
# zde se jen definuje, co se bude dělat při zavolání funkce s konkrétní hodnotou  
# zde se tedy zatím nedělá nic  
# definice funkce musí vždy předcházet jejímu volání  
def MojeFunkce(x):  
    return x*x - 2**x  
  
# volání funkce, zde se příkazy z definice funkce opravdu provádějí pro x = 3  
# tiskneme funkční hodnotu  
print(MojeFunkce(3))
```

Výstup programu:  
1

```
# tisk zadaného b a funkční hodnoty v bodě b  
def MojeFunkce(x):  
    return x*x - 2**x  
  
b = 5  
print(b, MojeFunkce(b))
```

Výstup programu:  
5 -7

```
# Tabulka hodnot funkce MojeFunkce() od -4 do 4  
def MojeFunkce(x):  
    return x*x - 2**x  
  
for u in range(-4, 5):  
    print(u, MojeFunkce(u))
```

Výstup:  
-4 15.9375  
-3 8.875  
-2 3.75  
-1 0.5  
0 -1  
1 -1  
2 0  
3 1  
4 0

```
# definujeme si vlastní funkci, která vypisuje druhé mocniny od 1 do N  
# zde se jen definuje, nedělá se nic  
def NaDruhou(N):  
    """Tabulka druhých mocnin do N."""  
    for k in range(1, N+1):  
        print(k, k*k)
```

```
# volání funkce, zde se příkazy z funkce opravdu provádějí  
NaDruhou(5)
```

Výstup programu:  
1 1  
2 4  
3 9  
4 16  
5 25

## Příklad – dělitelé

```
# Vypisují se dělitelé zadaného přirozeného čísla. Očekává se zadání přirozeného čísla n > 1.
n = int(input("Zadejte číslo, jehož dělitelé mají být vypsáni: ") )

for k in range(1, n+1):
    if n % k == 0: # je-li zbytek po dělení číslem k nulový, tak je k dělitel a tiskneme jej
        print(k, end=" ")

# při potřebě prošetřit několik konkrétních hodnot vnoříme for do tohoto cyklu for (pozor na odsazení):
for n in [98, 99, 101, 102, 998, 999, 1001, 1002]:
    print(n, end="\t")
    for k in range(1, n+1):
        if n % k == 0: # je-li zbytek po dělení číslem k nulový, tak je k dělitel a tiskneme jej
            print(k, end=" ")
    print() # vynechat řádek po vypsání dělitelích
```

Výstup programu:

```
98  1 2 7 14 49 98
99  1 3 9 11 33 99
101 1 101
102 1 2 3 6 17 34 51 102
998 1 2 499 998
999 1 3 9 27 37 111 333 999
1001 1 7 11 13 77 91 143 1001
1002 1 2 3 6 167 334 501 1002
```

---

Pokud několik řádků kódu:

- je třeba opakovaně využívat,
- má jasný smysl i samostatně,

tak by se tato část kódu měla stát funkcí.

Program se tím výrazně zpřehlední a snáze se udržuje.

*# funkce, která vypisuje všechny dělitele čísla n*

```
def Vypis_delitelu(n):
    print(n, end="\t")

    for k in range(1, n+1):
        if n % k == 0:
            print(k, end=" ")
    print()
```

```
for číslo in [12, 360, 20, 17]:
    Vypis_delitelu(číslo)
```

Výstup programu:

```
12  1 2 3 4 6 12
360 1 2 3 4 5 6 8 9 10 12 15 18 20 24 30 36 40 45 60 72 90 120 180 360
20  1 2 4 5 10 20
17  1 17
```

*# výpis dělitelů zadaného čísla – další možnost použití definované funkce*

```
n = int(input("Zadejte další číslo, jehož dělitelé mají být vypsáni: "))
Vypis_delitelu(n) # volání funkce
```

## Příklad – odmocniny

Výpočet odmocniny čísla  $x$  mezopotámským postupem,  
tj. pomocí rekurentně zadané posloupnosti  $a_{n+1} = (x + a_n^2) / (2 \cdot a_n)$

```
x = 5      # číslo, jehož odmocninu počítáme
N = 15     # počet iterací při výpočtu odmocniny (pro větší přesnost je potřeba více iterací)

a = x      # první člen posloupnosti

for i in range(1, N+1):
    a = (x + a*a) / (2*a)    # výpočet n-tého členu posloupnosti

print("Odmocnina čísla", x, "je", a)
```

Výstup programu:  
Odmocnina čísla 5 je 2.23606797749979

## Tabulka odmocnin počítaná mezopotámským postupem – funkce

$N$  – počet iterací při výpočtu odmocniny (pro větší přesnost je potřeba více iterací)  
 $N$  je inicializováno, takže je to nepovinný parametr  
nebude-li  $N$  při volání funkce zadáno, tak se použije defaultní hodnota 15

```
def Mezop_odmoc(x, N=15):
    a = x      # první člen posloupnosti

    for i in range(1, N+1):
        a = (x + a*a) / (2*a)    # výpočet n-tého členu posloupnosti

    return a

print("Číslo  Odmocnina")

for x in range(1, 6):
    print("{}\t{}".format(x, Mezop_odmoc(x)) )
```

Výstup programu:  
Číslo Odmocnina  
1 1.0  
2 1.414213562373095  
3 1.7320508075688772  
4 2.0  
5 2.23606797749979

*# testování vlivu počtu iterací na přesnost*  
print("Odmoc(5) počítaná pomocí n iterací:")

```
for n in range(1, 9):
    print("{}\t{}".format(n, Mezop_odmoc(5,n)) )
```

Výstup programu:  
Odmoc(5) počítaná pomocí n iterací:  
1 3.0  
2 2.3333333333333335  
3 2.2380952380952386  
4 2.2360688956433634  
5 2.236067977499978  
6 2.2360679774997894  
7 2.23606797749979  
8 2.23606797749979

## if – automatické slovní hodnocení na základě známky funkce nevracející a vracející hodnotu

```
def Hodnocení_print(n):  
    if n == 1:  
        print('výborně')  
    elif n == 2:  
        print('chvalitebně')  
    elif n == 3:  
        print('dobře')  
    elif n == 4:  
        print('dostatečně')  
    elif n == 5:  
        print('nedostatečně')  
    else:  
        print('takovou známku nemáme')
```

```
známka = int( input('Zadejte známku: ') )  
print(známka, end='\t')  
Hodnocení_print(známka)
```

*Výstup programu:*  
Zadejte známku: 1  
1      výborně

### Modifikace – funkce vracející řetězec (return místo print)

```
def Hodnocení_return(n):  
    if n == 1:  
        return 'výborně'  
    elif n == 2:  
        return 'chvalitebně'  
    elif n == 3:  
        return 'dobře'  
    elif n == 4:  
        return 'dostatečně'  
    elif n == 5:  
        return 'nedostatečně'  
    else:  
        return 'takovou známku nemáme'
```

```
známka = int( input('Zadejte známku: ') )  
print( známka, Hodnocení_return(známka) )
```

*Výstup programu:*  
Zadejte známku: 1  
1      výborně

## Zápis do souboru

```
# otevřeme soubor
můj_soubor = open("Můj název souboru.txt", "w")

i = 12

# zapisujeme do souboru
můj_soubor.write(str(i) + '\t' + "Ahoj, zapisuji do souboru." + '\n')
můj_soubor.write("A ještě něco na další řádek.")

# zavřeme soubor
můj_soubor.close()
```

.write()

- umí zapisovat do souboru pouze řetězec, tj. čísla je třeba konvertovat na řetězce pomocí funkce str()
- nepřidává narozdíl od print() odřádkování (proto je třeba přidávat \n)
- \t - tabulátor, \n - nový řádek

funkce open() otevře soubor s názvem uvedeným jako první parametr  
druhý parametr open():

"w" – otevření souboru pro zápis (write)

"r" – otevření souboru pro čtení (read)

"a" – otevření souboru pro přidávání dalších dat (append)

*# zápis do souboru pomocí spojení řetězců (primitivní postup)*

```
import math # importování knihovny matematických funkcí, v níž je funkce sqrt – odmocnina
```

```
f = open('NaDruhou.txt', 'w')

for i in range(1, 11):
    f.write(str(i) + '\t' + str(i*i) + '\t' + str(math.sqrt(i)) + '\n')

f.close()
```

*# tentýž program, využití formátovaného řetězce (lepší postup)*

```
import math

f = open('NaDruhou1.txt', 'w')

for i in range(1, 11):
    řádek = "{}\t{}\t{}\n".format(i, i*i, math.sqrt(i))
    f.write(řádek)

f.close()
```

## while – zatímco

```
a = 1
```

```
while a < 5:           # dokud bude a < 5, budou se příkazy provádět
    print(a)
    a = a + 1
```

Výstup programu:

```
1
2
3
4
```

## Hra – hádání čísla

*# Tipujeme číslo, dokud (while) jej neuhodneme.*

```
n = int( input("Tipněte si číslo od 1 do 5:  ") )
while n != 2:           # != nerovná se
    print("Neuhodli jste...")
    n = int( input("Tipněte si číslo od 1 do 5:  ") )
print("Ano, je to číslo 2.")
```

Výstup programu:

```
Tipněte si číslo od 1 do 5:  3
Neuhodli jste...
Tipněte si číslo od 1 do 5:  1
Neuhodli jste...
Tipněte si číslo od 1 do 5:  4
Neuhodli jste...
Tipněte si číslo od 1 do 5:  2
Ano, je to číslo 2.
```

## Nekonečná smyčka

```
# tisknou se postupně čísla od 1 do nekonečna
# podmínka ve while se nemění, je vždy splněna
a = 0
while True:
    print(a, end=" ")
    a = a + 1
```

## Malá násobilka – opakování cyklu for

```
print("Malá násobilka")
for i in range(1,11):
    for j in range(1,11):
        print(i, ".", j, "=", i*j)
    print()
```

## Rozklad na prvočísla

Vypisuje se rozklad zadaného přirozeného čísla na prvočísla  
očekává se zadání přirozeného čísla  $n > 1$

```
# funkce tiskne prvočísla z rozkladu n na prvočísla
def PrvociselnyRozklad(n):
    print("{} - rozklad na prvočísla:\t".format(n), end="")
    i = 2
    while n > 1:
        if n % i == 0:           # % zbytek po dělení
            print(i, end=" ")
            n = n // i          # // celočíselný podíl
        else:
            i = i + 1
    print()                    # vynechat řádek po vypsání prvočísel

# výpis prvočísel z rozkladu k na prvočísla
k = int(input("Zadejte číslo, jež rozložíme na prvočísla: "))

PrvociselnyRozklad(k)
```

*Výstup programu:*

Zadejte číslo, jež rozložíme na prvočísla: 360  
360 - rozklad na prvočísla: 2 2 2 3 3 5

*# při potřebě prošetřit několik konkrétních hodnot:*

```
for k in [98, 99, 101, 102, 998, 999, 1001, 1002]:
    PrvociselnyRozklad(k)
```

*Výstup:*

```
98 - rozklad na prvočísla: 2 7 7
99 - rozklad na prvočísla: 3 3 11
101 - rozklad na prvočísla: 101
102 - rozklad na prvočísla: 2 3 17
998 - rozklad na prvočísla: 2 499
999 - rozklad na prvočísla: 3 3 3 37
1001 - rozklad na prvočísla: 7 11 13
1002 - rozklad na prvočísla: 2 3 167
```

Možnost vylepšení:

funkce by vrátila pouze hodnoty

a je pak na uživateli,

v jaké podobě tyto hodnoty vytiskne, případně použije v dalších výpočtech.

## Základy práce s řetězci

```
s = "Totó je můj první řetězec v Pythonu."  
print(s) # vypíše řetězec s  
  
# indexuje se od nuly  
  
# tisk prvních 3 znaků řetězce (od 0 do 2) Tot  
print(s[:3])  
  
# tisk 2. až 3. znaku ot  
print(s[1:3])  
  
# tisk od 9. znaku až do konce můj první řetězec v Pythonu.  
print(s[8:])  
  
# tisk posledních 3 znaků nu.  
print(s[-3:])  
  
# spojování řetězců  
a = "matematická"  
b = "analýza"  
print(a + b) # matematickáanalýza  
print(a + " " + b) # matematická analýza  
  
c = a + " " + b  
print(c) # matematická analýza  
print(len(c)) # délka řetězce: 19
```

## Seznamy

```
# vektor 5 nul: [0, 0, 0, 0, 0]  
a = [0 for n in range(5)]  
print(a)  
  
# vektor 5 nul: [0, 0, 0, 0, 0]  
a = [0] * 5  
print(a)  
  
a[1] = 5  
print(a) # [0, 5, 0, 0, 0]  
  
# přidávání prvků do seznamu  
v = []  
for i in range(1, 7):  
    v.append(i)  
print(v)  
  
Výstup:  
[1, 2, 3, 4, 5, 6]
```

## Rozklad na prvočísla – seznam

očekává se zadání přirozeného čísla  $n > 1$   
nová verze s použitím funkce vracející seznam

funkce prvočísla netiskne, ale vrátí hodnoty  
je pak na uživateli, v jaké podobě tyto hodnoty vytiskne, případně je použije v dalších výpočtech

*# funkce vracející vektor prvočísel z rozkladu čísla n na prvočísla*

```
def PrvociselnyRozklad(n):  
    i = 2  
    P = []
```

```
    while n > 1:  
        if n % i == 0:  
            P.append(i)  
            n = n // i  
        else:  
            i = i + 1
```

```
    return P
```

*# při potřebě prošetřit několik konkrétních hodnot:*

```
for n in [98, 99, 101, 102, 998, 999, 1001, 1002]:  
    print("{} - rozklad na prvočísla: {}".format(n, PrvociselnyRozklad(n)))
```

Výstup programu:

```
98 - rozklad na prvočísla: [2, 7, 7]  
99 - rozklad na prvočísla: [3, 3, 11]  
101 - rozklad na prvočísla: [101]  
102 - rozklad na prvočísla: [2, 3, 17]  
998 - rozklad na prvočísla: [2, 499]  
999 - rozklad na prvočísla: [3, 3, 3, 37]  
1001 - rozklad na prvočísla: [7, 11, 13]  
1002 - rozklad na prvočísla: [2, 3, 167]
```

*# rozklad Fermatova čísla F5*

```
n = 2**2**5 + 1  
prv = PrvociselnyRozklad(n) # volání funkce, prv bude vektor  
print("Rozklad F5: ", prv)
```

Výstup programu:

```
Rozklad F5: [641, 6700417]
```

*# jiné využití vypočteného rozkladu na prvočísla:*

```
print("Všechna prvočísla < 100: ")
```

```
for n in range(2, 100):  
    rozklad = PrvociselnyRozklad(n)  
    if len(rozklad) == 1: # je-li délka seznamu == 1 (tj. obsahuje-li rozklad pouze samotné n)  
        print(n, end=" ")
```

Výstup programu:

Všechna prvočísla < 100:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

## Faktoriál

funkce vracující  $n!$  – naprogramována různými způsoby  
porovnání efektivity

*# faktoriál klasicky – for*

```
def faktorial_for(n):  
    f = 1  
    for i in range(1, n+1):  
        f = f * i  
    return f
```

*# faktoriál – while*

```
def faktorial_while(n):  
    f = 1  
    i = 0  
    while i < n:  
        i = i + 1  
        f = f * i  
    return f
```

*# faktoriál pomocí rekurze – funkce volá sama sebe*

```
def faktorial_rekurze(n):  
    if n < 2:  
        return 1  
    else:  
        return n * faktorial_rekurze(n - 1)
```

*# šlo by také:*

```
def faktorial_rekurze_nula(n):  
    if n <= 0:  
        return 1  
    else:  
        return n * faktorial_rekurze_nula(n - 1)
```

*# stručnější zápis, je i rychlejší*

```
def faktorial_rekurze_stručně(n):  
    return 1 if n < 2 else n * faktorial_rekurze_stručně(n - 1)
```

```
import time  
from sys import setrecursionlimit  
setrecursionlimit(10**5)
```

*n = 32000 # budeme počítat n!*

```
čas1_for = time.time()  
f = faktorial_for(n)  
čas2_for = time.time()  
print("for:\t", čas2_for - čas1_for)  
čas1_while = time.time()  
f = faktorial_while(n)  
čas2_while = time.time()  
print("while:\t", čas2_while - čas1_while)
```

```
čas1_rekurze = time.time()  
f = faktorial_rekurze(n)  
čas2_rekurze = time.time()  
print("rekurze:\t", čas2_rekurze - čas1_rekurze)
```

```
čas1_rekurze_stručně = time.time()  
f = faktorial_rekurze_stručně(n)  
čas2_rekurze_stručně = time.time()  
print("rekurze-stručně:\t", čas2_rekurze_stručně - čas1_rekurze_stručně)
```

*Výstup programu:*

```
for:      0.26843905448913574  
while:    0.2526528835296631  
rekurze:  0.29667210578918457  
rekurze-stručně:  0.2763817310333252
```



## Numerická integrace

```
import math

print("Integrál sinu od a do b obdélníkovou metodou.")

a = 0
b = math.pi
N = 10**6      # interval <a, b> dělíme na N stejných dílů

Delta = (b - a) / N
integ = 0

# sčítáme obsahy obdélníků,
# základna = Delta, výška = funkční hodnota ve středu dělicího intervalu
for k in range(0, N):
    integ += Delta * math.sin(Delta/2 + k*Delta)

print("integrál od", a, "do", b, "je roven: ", integ)
```

Výstup:

Integrál sinu od a do b obdélníkovou metodou.  
integrál od 0 do 3.141592653589793 je roven: 2.00000000000008424

*# Výpočet integrálu f od a do b  
# obdélníková metoda*

```
import math

def f(x):
    return (1 - x*x)**(1/2)

def g(x):
    return math.exp(math.sin(x)) * math.cos(x)

def h(x):
    return x * x

def integral(f, a, b, N=10**6):
    Delta = (b - a) / N
    integ = 0
    for k in range(N):
        integ += Delta * f(Delta/2 + k*Delta)
    return integ

# 4 * obsah jednotkového čtvrtkruhu
print(4 * integral(f, 0, 1))

print(integral(g, math.pi, 2*math.pi))

print(integral(h, 0, 1))
```

Výstup:

3.1415926539343633  
-8.512342306726817e-17  
0.3333333333332426

## Výpočet součtu prvních n členů dané řady

*# e = exp(1) = 1 + 1/1! + 1/2! + 1/3! + 1/4! + ...*

```
print("Počítáme e pomocí Taylorova rozvoje funkce exp x")
```

```
člen = 1
faktorial = 1
pocet_scitanych_clenu = 15
```

```
for i in range(1, pocet_scitanych_clenu + 1):
    faktorial *= i
    člen += 1 / faktorial
    print(i, člen)
```

*Výstup:*

Počítáme e pomocí Taylorova rozvoje funkce exp x

```
1 2.0
2 2.5
3 2.6666666666666665
4 2.7083333333333333
5 2.7166666666666663
6 2.7180555555555554
7 2.7182539682539684
8 2.71827876984127
9 2.7182815255731922
10 2.7182818011463845
11 2.718281826198493
12 2.7182818282861687
13 2.7182818284467594
14 2.71828182845823
15 2.718281828458995
```

*# ln 2 = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + ...*

```
s = 0
```

```
z = -1 # zajistí střídání znaménka
```

```
for n in range(1, 10**6 + 1):
    z *= -1
    s += z / n # s += (-1)**(n+1) / n by bylo mnohem pomalejší
```

```
print("součet členů řady: ", s)
```

```
from math import log # jen pro kontrolu
```

```
print("pro kontrolu: ln 2 =", log(2))
```

*Výstup:*

```
součet členů řady: 0.6931476805552527
pro kontrolu: ln 2 = 0.6931471805599453
```

## Výpočet hodnot funkce sinus s libovolnou přesností pomocí Taylorova rozvoje

```
"""
```

Funkce vracející sinus x, kde x je v radiánech.

Sinus se počítá pomocí Taylorova rozvoje se středem 0:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Taylorův rozvoj funguje nejlépe pro x blízka nule.

V případě větších x se doporučuje první vypočíst  $x = x \% (2 * \pi)$ .

Užití:

```
alfa = decimal.Decimal('0.3')
```

```
print(sin(alfa))
```

```
"""
```

```
import decimal
```

```
def sin(x, presnost=28):
```

```
    decimal.getcontext().prec = presnost + 2
```

```
    i, zbytek, soucet = 1, 0, x
```

```
    faktorial, citatel, znamenko = 1, x, 1
```

```
    while soucet != zbytek:
```

```
        zbytek = soucet
```

```
        i += 2
```

```
        faktorial *= i * (i-1)
```

```
        citatel *= x * x
```

```
        znamenko *= -1
```

```
        soucet += citatel / faktorial * znamenko
```

```
    decimal.getcontext().prec -= 2
```

```
    return +soucet # unární plus aktivuje nově nastavenou přesnost
```

```
alfa = decimal.Decimal('1')
```

```
print(sin(alfa))
```

```
print(sin(alfa, 50))
```

```
# pro kontrolu - vestavěná hodnota sinu:
```

```
from math import sin
```

```
print(sin(1))
```

Výstup:

```
0.8414709848078965066525023216
```

```
0.84147098480789650665250232163029899962256306079837
```

```
0.8414709848078965
```

## Řešení rovnice $f(x) = 0$ metodou půlení intervalu

""" Řešení rovnice  $f(x) = 0$  metodou půlení intervalu

předpokládá se zadání dvou čísel, mezi kterými má rovnice  $f(x) = 0$  právě jeden kořen

1. program zkontroluje, zda mezi zadanými čísly leží kořen
2. provádí se půlení intervalu, dokud není jeho délka menší, než zadaná přesnost"""

*# zadání intervalu, v němž budeme hledat kořen*

a, b = 0, -1

presnost = 2 \* 10\*\*(-16)

**import** math

*# definice funkce f*

**def** f(x):  
 **return** x\*x - 2\*\*x *#x - math.cos(x) # x\*x - 2\*\*x*

*# funkce, která zúží interval*

**def** zuzit\_interval(v):  
 a = v[0]  
 b = v[1]  
 c = (a + b) / 2  
  
 **if** f(a) == 0:  
 **return** [a]  
 **if** f(b) == 0:  
 **return** [b]  
 **if** f(c) == 0: *# lze zvolit toleranci: abs(f(c)) < presnost*  
 **return** [c]  
 **if** (f(a) < 0 **and** f(c) > 0) **or** (f(a) > 0 **and** f(c) < 0):  
 **return** [a, c]  
 **if** (f(b) < 0 **and** f(c) > 0) **or** (f(b) > 0 **and** f(c) < 0):  
 **return** [c, b]  
 **else**:  
 **return** "kořen nenalezen"

interval = [a, b]

pocet = 0

*# hledání kořene*

**while** len(interval) == 2 **and** abs(interval[1] - interval[0]) > presnost:  
 interval = zuzit\_interval(interval)  
 pocet = pocet + 1

*# použijeme raději výpis hodnot pod sebe - usnadňuje srovnání*

**if** len(interval) == 2:  
 **for** x **in** interval:  
 **print**(x)

**else**:  
 **print**(interval)

**print**("počet půlení:", pocet)

Výstup:

-0.766664695962123

-0.7666646959621232

počet půlení: 53

## Generování pythagorejských trojic

```
"""
Vypisujeme do tabulky pythagorejské trojice,
tj. trojice [a, b, c] splňující  $a^2 + b^2 = c^2$ 
jednoduchá verze, trojice generované vzorci:
 $a = u^2 - v^2$ 
 $b = 2*u*v$ 
 $c = u^2 + v^2$ 
přičemž  $0 < v < u$ 
"""

N = 5      # čísla u, v se volí až do N

# kolik for, tolikátice se volí
for u in range(1, N+1):
    for v in range(1, u):
        print("{}\t{}\t{}".format(u*u - v*v, 2*u*v, u*u + v*v))
```

Výstup:

3	4	5
8	6	10
5	12	13
15	8	17
12	16	20
7	24	25
24	10	26
21	20	29
16	30	34
9	40	41

## Inspirace Archimédovou aproximací $\pi$

```
"""
Aproximujeme pí pomocí
obvodů vepsaných pravidelných n-úhelníků:
o_n = 2 * r * n * sin (180° / n)

porovnáním s obvodem kruhu: o = 2 * r * pi
dostáváme aproximaci pí:

pi_n = n * sin (180° / n)
"""

import math

print("Aproximace pí pomocí obvodů vepsaných pravidelných n-úhelníků:")

for k in range(1, 6):
    n = 10 ** k
    VnitřníUhel = math.pi / n
    pi_n = n * math.sin(VnitřníUhel)
    print(n, "\t", pi_n)
```

*Výstup:*

```
Aproximace pí pomocí obvodů vepsaných pravidelných n-úhelníků:
10          3.090169943749474
100         3.141075907812829
1000        3.1415874858795636
10000       3.141592601912665
100000      3.1415926530730216
```

## Různé způsoby výpočtu binomických koeficientů

*# POZOR - zde je správně jen prvních 15 cifer, pak chyba zaokrouhlení float*

*# navíc kvůli float: maximálně lze kombin\_mala(1020, 510)*

*# kombinační čísla n nad k*

```
def kombin_mala(n, k):  
    komb = 1  
    for i in range(1, k+1):  
        komb = komb * (n-i+1) / i  
    return int(komb)
```

*# kombinační čísla n nad k - spolehlivá funkce i pro velké vstupní hodnoty*

```
def kombin_velka(n, k):  
    komb = 1  
    for i in range(1, k+1):  
        komb = komb * (n-i+1)  
    for i in range(1, k+1):  
        komb = komb // i  
    return komb
```

*# kombinační čísla pomocí zlomků - eliminace chyby zaokrouhlení*

*# funguje tedy spolehlivě i pro obrovská čísla*

```
from fractions import Fraction
```

```
def kombin_frac(n, k):  
    nk = Fraction(1,1)  
    for i in range(1, k+1):  
        nk *= Fraction(n-i+1, i)  
    return nk
```

*# kombinační čísla - počítána pomocí rekurze*

```
def kombin_rekurze(n, k):  
    if n < k:  
        return 0  
    if n < 2 and k < 2:  
        return 1  
    if k == 0:  
        return 1  
    else:  
        return kombin_rekurze(n-1,k-1) + kombin_rekurze(n-1,k)
```