Ciklai

`for` ciklas

`for` ciklas skirtas pasiimti elementus vieną po kito iš sąrašo (eilutės). Kitaip tariant, `for` paima iš sąrašo elementą, atlieka nurodytus veiksmus su elementu, ir eina prie kito elemento. Elementai imami iš eilės.

Sintaksės pvz.

```
sr = [1,2,3]
for s in sr:
  print(s) # s - ciklo
   ↪  kintamasis
```

`for` ciklas su `range()`

```
for skaicius in range(0, 10):
    print(skaicius)
```

`range(a,b, step)` - skaičių sekos generatorius. a - pradinis skaičius, nuo kiek skaičiuojama, b - iki kiek skaičiuojama, $step$ - kas kiek skaičiuojama. `range()` funkcijos rezultatą galima įsivaizduoti kaip virtualų sąrašą.

for ciklo pavyzdys

```
vardu_sarasas = ['Jonas', 'Petras', 'Ona', 'Agota']
for vardas in vardu_sarasas:
    print(vardas)
```

Iteracija per simbolius

```
txt = "sako, kad Delfi.lt yra populiariausias portalas."
for letter in txt:
    print(letter)
```

`for` ciklas

VILNIUS CODING SCHOOL

Iteravimas per žodyną

```
for k in zod:
    print(k)
```

Kuo skirsis?

```
for k in zod.keys():
    print(k)
```

Iteravimas per vertes

```
for k in zod.values():
    print(k)
```

📌 `for` sintaksė nesiskirs nuo iteravimo per sąrašą ir iteravimo per eilutę ar rinkinį.

VILNIUS CODING SCHOOL

enumerate()

Jei reikia iteruoti per sąrašą, ir reikia ir sąrašo elemento, ir jo pozicijos, gali praversti `enumerate()` funkcija:

```
t = "labas rytas"
for idx, letter in enumerate(t):
    print(idx, letter)
```

Kodėl čia netinka string metodas `.index()`? 🤔

zip()

Jei reikia iteruoti iškart per kelis sąrašus, verta sujungti:

```
a = [-1,0,1,2,3,4,5,6,7,8]
b = [7,8,9,10,11,12,13,14,15,16,17,18,19]
c = []
for a1, b1 in zip(a,b, strict=False):
    c.append(a1*b1)
print(a, len(a), b, len(b), c, len(c))
```

Užduotys

- Paprašykite vartotojo įvesti simbolį bei skaičių. Išveskite kvadratą, sudarytą iš įvesto simbolio ir kurio kraštinė būtų lygi įvestam skaičiui. Pvz.: įvesta `'@'` ir 5. Rezultatas:

```
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
```

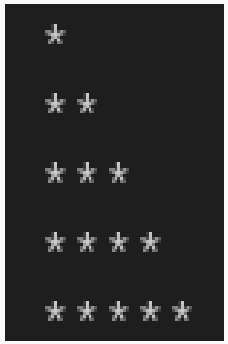
Užduotys

Duotas sąrašas:

```
miestai = ['Vilnius', 'Kaunas', 'Alytus', 'Rokiškis',
'Ūla', 'Mažeikiai', 'Akmena']
```

Spausdinkite tik tuos miestus, kurių pavadinimai ilgesni nei 6 simboliai.

Išspausdinkite ”pusinę” eglutę iš simbolio `'*'`:



Eglutės ilgį (eilučių kiekį) turi įvesti vartotojas. Eglutei naudojamą simbolį tegul įveda irgi vartotojas.

Užduotys

VILNIUS CODING SCHOOL

```
t = "Aš rytais mėgstu kavą su sumuštiniais ir arbatą"
```

- Išveskite po vieną simbolį iš teksto. Šalia nurodykite ir simbolio poziciją (indeksą) tekste.
- Išvesdami simbolį, spausdinkite jį viršutiniame registre, jei jis yra lyginėje pozicijoje. Kitu atveju - žemutiniame registre.
- Išveskite po vieną žodį.
- Išveskite tik tuos žodžius, kurie yra ilgesni nei nurodytas simbolių kiekis, ir savyje turi nurodytą tekstą. Ilgį ir tekstą nurodo vartotojas.
- Išveskite tekstą (visą vienu metu), kuriame būtų kas antras žodis parašytas didžiosiomis raidėmis.
- Išveskite tekstą su kas n-tuoju simboliu viršutiniame registre. n įveda vartotojas.

Užduotys

Duoti sąrašai:

```
a = [1,2,3,4,5]
b = [-7,-2,0,1,4]
```

- Suskaičiuokite:
  - $c = a + b$
  - $d = a \cdot b$
  - $e = \frac{a}{b}$
- Sugeneruokite skaičių sąrašą nuo n iki k, kas m. n, k, m įveda vartotojas.
  - Suraskite šio sąrašo skaičių sumą, aritmetinius ir geometrinius vidurkius.

Užuominos

`sum()` - suras sąrašo skaičių sumą. $A_{aritm.} = \frac{\sum_0^n a_k}{n}$,

$A_{geom.} = \sqrt[n]{\left(\prod_0^n a_k\right)}$

VILNIUS CODING SCHOOL

while ciklas

Naudojamas, kai reikia atlikti tam tikrus veiksmus, kol yra tenkinama kokia nors išanksinė sąlyga.
Šablonas:

```
k = ?
while loginė sąlyga su k:
    veiksmai
    # turi būti kodas, keičiantis sąlygoje esančią k vertę!
```

while ciklas

```
i = 0
while i < 10:
    print(i)
    i = i  + 1 # jei nebus šios eilutės - gausite amžiną ciklą!
```

`while` ciklas

Amžino `while` ciklo pavyzdys

```
quit = False
while not quit:
    q = input("Įveskite q")
    if q != 'q':
        continue
    else:
        quit = True
```

`continue`, `break` ir `pass` raktažodžiai

- `continue` - pereis prie kitos iteracijos, nevykdant likusio ciklo kodo dalies
- `break` - nutrauks ciklo vykdymą
- `pass` - ”Nieko nedaryti”. Jei pagal sintakę reikia įrašyti kodą, bet nieko nenorime atlikti. `pass` dažnai naudojamas kaip žymė vietai, kur bus ateities kodas (*placeholder for future code*)

`break` ir `continue` panaudojimas

```
for i in range(0,4):
    for k in range(0,11):
        if k == 5:
            continue
        if k == 9:
            break
        print(k, k**2)
    print('Iteracija (žingsnis): {}'.format(i))
```

Pilnas `for` bei `while` ciklo šablonas

VILNIUS CODING SCHOOL

```
i = [*range(0,5)]
for k in i:
    print(f'{k = }') #cikle atliekami veiksmai
else:
    print('for ended correctly') # else šaka nebūtina, joje rašomi
    ↪ veiskmai, atliekami po SĖKMINGO visų ciklo veiksmų užbaigimo.

r = 2
while r > 0:
    print(f'{r = }')
    r = r - 1
else:
    print('while ended correctly') # else šaka nebūtina, joje rašomi
    ↪ veiskmai, atliekami po SĖKMINGO visų ciklo veiksmų užbaigimo.
```

Užduotys su `while`

Pakartoti užduotį, esančią 79 skaidrėje, naudojant **while**.

- Parašyti amžiną `while` ciklą, kuris būtų nutraukiamas, jei vartotojas įveda Q simbolį. Simbolio registras neturi turėti reikšmės.

Užduotys

Duotas tekstas `t="Vikipedija yra universali, daugiakalbė interneto enciklopedija, kaip bendruomeninis projektas, pagal viki technologiją ir pamatinius principus kuriama daugybės savanorių bei išlaikoma iš paaukotų lėšų."`

Užduotys

- Išveskite tekstą su kas antra didžiąja raide.
- Išveskite tekstą su kas n-taja didžiąja raide. n įveda vartotojas.
- Išveskite tekstą su kas n-tuoju žodžiu, parašytu didžiosiomis raidėmis. n įveda vartotojas.
- Išveskite tekstą, kurio kiekvienas žodis prasidėtų didžiąja raide.

⚠️ - galite rinktis, ar `for`, ar `while` naudoti.

Klaidų vengimas, valdymas

`try except` šaka klaidų paieškai

VILNIUS CODING SCHOOL

Esant tikėtinai klaidos situacijai, naudojama `try` sakinys.

Toks sakinys veikia panašiai kaip `if` sakinys, tačiau čia nėra sąlygų tikrinimo.

Pilno `try` šablonas

```
try:
    pass #čia bus vykdomas pagrindinis kodas
except Exception as ex: #except šakų gali būti daugiau nei 1-na
    pass #čia bus vykdomas kodas, jei try šakoje KILS klaida
else: #nebūtinas
    pass #čia bus vykdomas kodas, jei NEKILS problemų
finally: #nebūtinas
    pass #kodas bus vykdomas nepaisant ar kilo problema, ar ne.
```

Galimi standartiniai klaidų tipai

Rekomendacijos:

- Nenaudoti bazinio `Exception` tipo - specifikuoti veiksmus pagal galimas problemas
- Jei galima užtikrinti su `if` konstrukcija, kad nesusidarytų sąlygos klaidai - naudokite `if`

Pavyzdžiai

Jei neturėtumėme klaidų valdymo

```
numbers = [-2,-1,0,1,2]
for n in numbers:
    print(5/n) # čia trečio žingsnio metu bus 5/0 - dalyba iš 0
```

Su try konstrukcija

```
numbers = [-2,-1,0,1,2]
for n in numbers:
    try:
        print(5/n) # čia trečio žingsnio metu bus 5/0 - dalyba iš
        ↪ 0
    except:
        print('dalyba iš 0! Bet programa veiks toliau')
```

Užduotys

Parašykite kodą, kuris paklaustų vartotojo skaičių A ir B, matematinio veiksmo ($+$, $-$, $/$, $*$), atliktų veiksmą, rezultatą išspausdintų. Pritaikykite `try` bloką, kad išvengtumėte tokių situacijų kaip:

- dalyba iš 0;
- vietoj skaičiaus tekstas;

Pritaikykite visas `try` bloko šakas.

Funkcijos

Funkcijos

Funkcija naudojama, kai reikia dažnai atlikti grupę veiksmų, kad jų nereiktų aprašyti kiekvienąkart po vieną veiksmą.

Funkcijos šablonas

```
def funkcijos_pavadinimas():
    veiksmai, kuriuos atliks funkcija
```

Į funkciją kreipiamasi (ji iškviečiama) užrašant jos pavadinimą: funkcijos_pavadinimas().

Funkcijos pavyzdys

```
def fn_name():
    print("Aš esu funkcijoje")
#iškvietimas:
fn_name()
```

Funkcijų rūšys

VILNIUS CODING SCHOOL

- Funkcija, kuriai nieko neperduodama, kuri nieko negrąžina:

```
def funkcija():
    print('Esu paprasta funkcija')
```

- Funkcija su parametru, nieko negrąžina

```
def funkcija(parametras):
    print('Man buvo nurodyta', parametras)
funkcija(5) # kreipimasis į funkciją - būtina nurodyti
 ↪  parametrą
```

Funkcijų rūšys

VILNIUS CODING SCHOOL

- Funkcija su parametrais (gali būti jų bet kiek, čia rodoma 3)

```
def funkcija(par1, par2, par3):
    print('Pateikti parametrai', par1, par2, par3)
funkcija(1,2,'trečias parametras')
funkcija(par3=2, par1=15, par2='OMG')
```

- Funkcija su parametrais ir numatytosiomis reikšmėmis

```
def funkcija(par1, par2=10):
    print('Pateikti parametrai:', par1, par2)
funkcija(1,15)
funkcija(2) #viskas O.K., Python žinos, kad par2=10
```

Funkcijų rūšys

- Funkcijos, grąžinančios atsakymą

Grąžinimo pavyzdys

```
def funkcija(par1=1):
    return par1*2 #ši funkcija padaugins parametrą par1 ir
     ↪  grąžins atsakymą mums į pagrindinę programą
ats = funkcija(5)
```

Jei funkcija turi raktažodį `return` - būtina prieš kreipimąsi į funkciją užrašyti priskyrimą kintamajam, kitaip atsakymas bus prarastas.

Kintamųjų išpakavimas (*unpacking*)

VILNIUS CODING SCHOOL

- Kintamųjų išpakavimas - kai Python automatiškai suskaido metodo ar funkcijos atsakymą ir priskiria nurodytiems kintamiesiems.

```
def fn():
    return 3 , 5
a = fn()
print(type(a))
print(a[0])
print(a[1])
```

```
def fn():
    return 3 , 5
a, b = fn()
print(a)
print(b)
```

```
t ="0.55;0.66"
x, y = t.split(';')
```

```
def fn():
    return 1,2,3,4,5
a, *b, c = fn()
*raides, = 'tekstas'
```

map() funkcija

```
map(fun, iter) #sintaksės pavyzdys
```

- `map()` funkcija pritaiko nurodytą išraišką (funkciją *fun*) sąrašui (eilutei, iteruojamam objektui). Naudinga, kai reikia atlikti sudėtingus veiksmus su kiekvienu sąrašo elementu.

⚠️ Rezultatas - **map** objektas. Norint gauti normalų sąrašą (eilutę, ...), **map** objektą reikia pateikti `list()` ar `tuple()` funkcijai:

```
it = [1,2,3,4,5]
def fn(x):
    return x*2
result = map(fn, it)
print(list(result))
```

Uždaviniai

1. Parašyti funkciją, kuri priima sąrašą ir grąžina naują sąrašą su paskutiniu ir pirmu elementais iš pirmojo sąrašo. PVZ: [1,2,3,4,5,6,7,8,9] ats: [1,9]
2. Parašyti funkciją greičiui tikrinti: Funkcija paima argumentą - automobilio greitį. Jei greitis 50 ar mažesnis, funkcija grąžina ”Ok”, jei greitis didesnis nei 50 - už kiekvieną 5km/h greičio viršijimą duodamas baudos taškas (pvz: greitis 70, atsakymas - 4) Jei surenkami 8 taškai ir daugiau - funkcija ne tik grąžina taškų sumą, bet ir priduria, jog vairuotojo teisės atimamos.

Uždaviniai

3. Parašyti funkciją, kuri priima tekstą ir grąžina atgal tekstą su apkeistom pirma ir paskutine raidėmis. PVZ: Rytas → sytaR.
4. Parašyti funkciją, kuri paima ir parašo žodį iš kitos pusės. PVZ.: Dangus → sugnaD.
5. Parašyti funkciją, kurioje sugeneruojamos dvi skaičių sekos (jos gali būti skirtingo dydžio), programa turi grąžinti sąrašą skaičių, kurie sutampa tarp dviejų sugeneruotu sąrašų, galutiniame sąraše neturi būti skaičių dublikatų. PVZ: x =[1, 1, 2, 3, 1] y = [1, 3, 5, 6] atsakymas = [1,3]

Funkcijos su n argumentų

VILNIUS CODING SCHOOL

Funkcija gali turėti nenurodytą argumentų kiekį

```
def fn_multi_args(*args):
    print(args) #pats args
    print(type(args)) # jo tipas
    print(sum(args)) #veiksmas su args
fn_multiargs(1,2,3,4,5,6,7,8)
```

Po *args būtinai eina tik raktažodžiai (*keyworded arguments*)

```
def fn_multiargs(arg1, *args, arg2):
    print(arg1, arg2)
    print(args)
fn_multiargs(1,2,3,4) #neveiks
fn_multiargs(1,2,3,arg2 = 4) #veiks, būtina nurodyti!
```

Funkcijos su n argumentų

Funkcija gali turėti nenurodytą kiekį vardinių argumentų (*keyworded arguments*)

```
def fn_multi_args(**kwargs): # ** žymi raktažodžius
    print(type(kwargs))
    print(kwargs)
fn_multi_args(k=1,k2=2,k3=3) # veiks
fn_multi_args(1,2,3) # neveiks, interpretuojama kaip poziciniai
 ↪  argumentai
```

```
def fn(a, b, *c, d, **d_):
    print(a, b)
    print(c)
    print(d)
    print(d_)
fn(1,2, d=5) #poziciniai ir vienas keyworded argumentas būtini
 ↪  nurodyti
```

Funkcijos su n argumentų

Funkcijos su visais argumentų tipais šablonas

```
def fn(a,b, *args, d, e, **kwargs):
    print("a, b - poziciniai, būtina nurodyti kreipiantis")
    print("args - nebūtina nurodyti")
    print("d, e - raktažodžiai, būtina nurodyti (nebent turi
     ↪  numatytąsias reikšmes)")
    print("kwargs - nebūtina nurodyti kreipiantis")
fn(1,2,d=3,e=4) #kaip matyti, nurodyti būtina pozicinius
 ↪  argumentus ir raktažodžius (jei jie neturi numatytųjų
 ↪  reikšmių)
```

Užduotys

- Parašykite funkciją, kuri suskaičiuotų aritmetinį arba geometrinį vidurkį iš pateiktų skaičių. Pateikiamų skaičių kiekis neapibrėžtas. Panaudokite raktažodžius, kad nurodytumėte, kokį vidurkį reikia apskaičiuoti.

Užuominos

`sum()` - suras sąrašo skaičių sumą. $A_{aritm.} = \frac{\sum_0^n a_k}{n}$,

$A_{geom.} = \sqrt[n]{\left(\prod_0^n a_k\right)}$

assert

`assert` raktažodis

Esama situacijų, kai reikia iškart patikrinti duomenų tinkamumą, ir jei jie netinkami, programa turi liautis veikti (Kodo testavimas (*debugging*)).

`assert` sintaksė

```
assert boolean, 'message'
```

Čia `boolean` - loginė išraiška, kurios netenkinus, rodoma `AssertionError` klaida, o `'message'` - klaidos pranešimas (nurodyti nebūtina).

```
a = 5
b = 0
print("a / b veiksmo rezultatas : ")
assert b != 0, "Dalyba iš nulio!!!"
print(a / b)
```

`assert` funkcijose

Patogu pačioje funkcijoje turėti tikrinimą, ar duomenys yra teisingi:

```
def area(length, width):
    assert length > 0 and width > 0, 'Matmenys turi būti
     ↪  teigiami'
    return length*width
print(area(5,6))
print(area(-5,6))
```

⚠️ Jei `assert` parodė klaidą, tai rodo ne funkcijos, kurioje yra `assert` raktažodis, neteisingumą, o prieš tai vykdomo kodo problemas! Prieš tai esantis kodas siunčia į jūsų funkciją neteisingus duomenis.

Užduotys

- Parašykite funkciją, kuri galėtų ištraukti nurodyto laipsnio šaknį iš nurodyto skaičiaus. Su `assert` užtikrinkite, jog jei nurodoma lyginio laipsnio šaknis, tai skaičius (iš kurio traukiama šaknis) turi būti teigiamas, kitu atveju rodykite `AssertionError` klaidą.

Šaknies traukimui naudokite tapatybę:

$$\sqrt[n]{x} = x^{\frac{1}{n}} \tag{1}$$

✔️ Jei *Python* ****** operatorius grąžina menamus skaičius, panaudoti `abs()` funkciją.

`lambda` funkcijos

Anoniminės (`lambda`) funkcijos

VILNIUS CODING SCHOOL

`lambda` funkcijos, arba kitaip anoniminės funkcijos, naudojamos ten, kur reikia trumpo kodo, kurio neverta aprašyti kaip funkcijos. `lambda` funkcijoje negalima naudoti jokių raktažodžių kaip `return`, `assert`, `pass`. `lambda` funkcija iš esmės visada yra viena trumpa išraiška (*single expression*).

`lambda` naudojimas

```
#normali funkcija:
def ret(x):
    return x
#ją atitinkanti lambda išraiška:
lambda x: x
```

⚠️ `lambda` galima priskirti pavadinimą:

```
ret = lambda x: x
ret(2)
```

Anoniminės (lambda) funkcijos

VILNIUS CODING SCHOOL

lambda naudojimas

```
str1 = 'Python yra Python.'
upper = lambda string: string.upper()
#pavadinimas = raktažodis kintamasis : veiksmas
print(upper(str1))
```

```
l = [1,2,3,4,5,6]
# raskime l**2:
print(list(map(lambda x: x**2, l)))
```

Nereikėjo deklaruoti naujos funkcijos, skirtingai nuo pavyzdžio (95) skaidrėje.