Prednáška č. 3

Zapúzdrenie, generalizácia, interface, refaktorizácia

O čom bude dnešná prednáška

- Dnes si na prednáške povieme pár slov o tom, ako písať funkcie, čo je to enkapsulácia a generalizácia.
- Taktiež si ukážeme, že písanie kódu je kreatívny proces a často sa vraciame k tomu, čo už funguje, aby sme to prípadne zlepšili (refaktorizácia)

Najprv program pre štvorec

- Na úvod urobíme program, ktorý pomocou modulu *turtle* vykreslí štvorec

```
import turtle

turtle.forward(100)
turtle.left(90)

turtle.forward(100)
turtle.left(90)

turtle.forward(100)
turtle.left(90)

turtle.forward(100)
turtle.left(90)
```

Vidíme, že v programe, ktorý sme spravili, sa nám 4-krát opakuje dvojica príkazov:

“choď vpred o 100 pixelov”
(turtle.forward(100))
“otoč sa vľavo o 90 stupňov”
(turtle.left(90))

Preto to vieme popísať aj pomocou for-cyklu:

```
import turtle

for i in range(4):
    turtle.forward(100)
    turtle.left(90)
```

Zapúzdrenie (enkapsulácia)

- Ak máme časť kódu, ktorá vykonáva nejakú činnosť, je vhodné vytvoriť z nej funkciu.
- Tento proces nazývame **enkapsulácia / zapúzdrenie (encapsulation).**
- Zapúzdrenie je proces, pri ktorom zapúzdrime (“zabalíme”) kód do funkcie.
- Následne môžeme v programe namiesto pôvodného kódu používať príslušnú funkciu, čo okrem iného zlepšuje čitateľnosť kódu.

Ak zapúzdrime kód, ktorý kreslí štvorec do funkcie *stvorec()*, dostaneme funkciu, ktorú môžeme následne použiť na vykreslenie štvorca.

Výhodou zapúzdrenia je navyše to, že ak sa vhodne zvolí samotný **identifikátor funkcie**, t.j. *stvorec,* vieme z neho vydedukovať, čo funkcia robí!

```
import turtle

def stvorec():
    for i in range(4):
        turtle.forward(100)
        turtle.left(90)

stvorec() #volanie funkcie stvorec
```

zapúzdrenie kódu kresliaceho štvorec do funkcie stvorec()

Zovšeobecnenie (generalizácia)

- Ak máme funkciu, ktorá realizuje nejakú činnosť, niekedy je vhodné zamyslieť sa, či nevieme vytvoriť jej **všeobecnejšiu verziu,** ktorá bude vedieť vykonávať danú činnosť všeobecnejšie.

- Tento proces nazývame **zovšeobecnenie / generalizácia (generalization).**

- Pri zovšeobecnení funkcie spravidla pridáme do hlavičky funkcie nový vstupný parameter pomocou ktorého vieme parametrizovať typ činnosti, ktorý robí daná funkcia. Zároveň vždy platí, že pre vhodnú voľbu tohto parametra vie funkcia realizovať aj pôvodnú činnosť!

- V predošlom príklade sme zostrojili funkciu, ktorá kreslí **štvorec so stranou fixnej dĺžky** 100 pixelov. Preto by sme sa mohli pokúsiť funkciu **generalizovať**, t.j. zostrojiť funkciu, ktorá bude vedieť kresliť štvorce **so stranami ľubovoľnej dĺžky**, pričom táto dĺžka bude **nový parameter funkcie**.

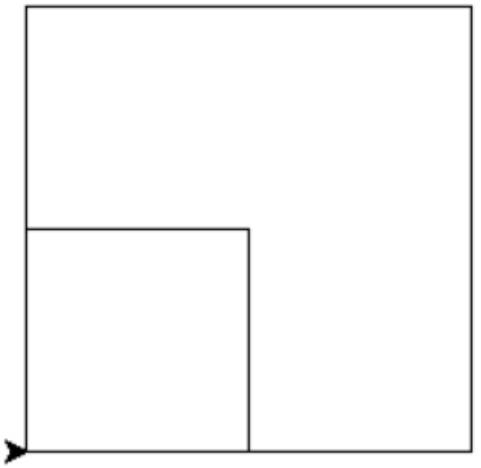
Zovšeobecníme predošlú funkciu *stvorec()* tak, sme pomocou jej vstupného parametra vedeli meniť veľkosť strany kresleného štvorca.

Pridáme teda do hlavičky parameter *dlzka_strany* a v tele funkcie tento parameter použijeme ako dĺžku jednotlivých strán štvorca. Všimnite si, že pre voľbu vstupu 100 sa funkcia *stvorec*(100) správa ako pôvodná funkcia, ktorú sme zovšeobecnili!

```
import turtle

def stvorec(dlzka_strany):
    for i in range(4):
        turtle.forward(dlzka_strany)
        turtle.left(90)

stvorec(100) #stvorec so stranou dlzky 100 pixelov
stvorec(200) #stvorec so stranou dlzky 200 pixelov
```

pridaný parameter

funkcia teraz pre rôzne vstupné argumenty kreslí rôzne štvorce



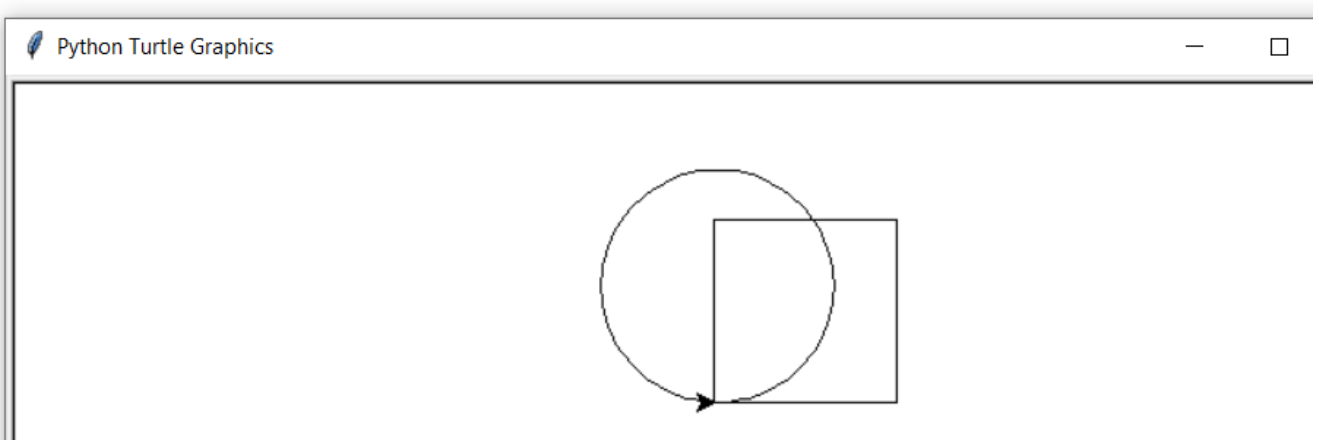
Funkciu *stvorec(dlzka_strany)* vieme zovšeobecniť ešte viac! Vytvoríme z nej funkciu *nuholnik(n, dlzka_strany)* ktorá bude kresliť pravidelný *n*-uholník s *n* uhlami a stranou dĺžky *dlzka_strany.* V tomto prípade však do kódu budeme musieť pridať výpočet uhla, o ktorý sa musí korytnačka vždy otočiť, aby bol n-uholník korektný.

Všimnite si, že pre voľbu *n* = 4 by sa funkcia správa ako pôvodná funkcia z predošlého slajdu, ktorú sme zovšeobecnili!

```
import turtle

def nuholnik(n, dlzka_strany):
    uhol = 360/n
    for i in range(n):
        turtle.forward(dlzka_strany)
        turtle.left(uhol)

nuholnik(4,100) #stvorec so stranou dlzky 100 pixelov
nuholnik(3,200) #rovnostranny trojuholnik so stranou dlzky 200 pixelov
```

Ako sme si povedali na minulej prednáške, pri volaní funkcie je nutná opatrnosť pri poradí argumentov!

```
import turtle

def nuholnik(n, dlzka_strany):
    uhol = 360/n
    for i in range(n):
        turtle.forward(dlzka_strany)
        turtle.left(uhol)

nuholnik(4,100) #stvoruholnik so stranou dlzky 100 pixelov
nuholnik(100,4) #stouholnik so stranou dlzky 4 pixely
```



Pomenované argumenty

V niektorých jazykoch – vrátane Pythonu – je možné pri volaní funkcie použiť tzv. pomenované argumenty (keyword arguments), vďaka ktorým je možné volať funkciu tak, že sa **nedodrží** poradie argumentov, ale **špecifikuje sa**, ktorý argument patrí ktorému vstupnému parametru.

To, ktorý argument patrí ktorému vstupnému parametru sa špecifikuje pri volaní funkcie tak, že sa priamo vo volaní uvedie pri jednotlivých argumentoch uvedie:

```
parameter = hodnota argumentu
```

Lepšie to je vidieť na príklade na ďalšom slajde

Hoci definícia funkcie *nuholnik(n, dlzka_strany)* uvažuje, že prvý parameter je počet uhlov a druhý parameter dĺžka strany, ak pri volaní použijeme **pomenované argumenty**, môžeme pri volaní ich poradie aj zameniť, teda:

nuholnik(4, 100) je volanie, ktoré vykreslí 4-uholník so stranou dĺžky 100
nuholnik(**dlzka_strany=100, n = 4**) je volanie, ktoré vykreslí 4-uholník so stranou dĺžky 100
nuholnik(100,4) je volanie, ktoré vykreslí 100-uholník so stranou dĺžky 4

```
import turtle

def nuholnik(n, dlzka_strany):
    uhol = 360/n
    for i in range(n):
        turtle.forward(dlzka_strany)
        turtle.left(uhol)

nuholnik(4,100) #stvoruholnik so stranou dlzky 100 pixelov
nuholnik(dlzka_strany = 100, n = 4) #tiez stvoruholnik so stranou dlzky 100 pixelov
nuholnik(100,4) #stouholnik so stranou dlzky 4 pixely
```

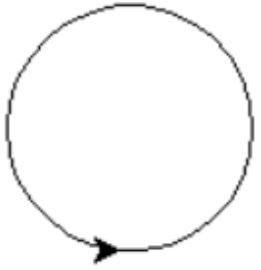
Využitie funkcií

Cieľom definície funkcií je mať ich k dispozícii pre ďalšie použitie. Dôležité je vedieť, čo sú jej vstupy/výstupy a čo funkcia robí, pretože potom viem navrhnúť program, ktorý bude funkciu využívať za nejakým účelom.

Napríklad ak by som chcel vykresliť kružnicu, viem použiť funkciu kresliacu n-uholník, pretože n-uholník s dostatočne veľkým počtom uhlov vyzerá ako kružnica.

Definujme teda ďalšiu funkciu *kruznica(polomer)*, ktorá vykreslí kružnicu s polomerom, ktorý predstavuje jej vstupný parameter. Táto funkcia vykreslí kružnicu tak, že ju aproximuje pomocou n-uholníka, pričom dĺžka strany n-uholníka bude zvolená tak, aby obvod n-uholníka približne korešpondoval s obvodom kružnice.

```python
import turtle

def nuholnik(n, dlzka_strany):
    uhol = 360/n
    for i in range(n):
        turtle.forward(dlzka_strany)
        turtle.left(uhol)

def kruznica(polomer):
    obvod_kruznice = 2 * 3.1415 * polomer
    n = 30 #na aproximaciu kruznice pouzijeme 30-uholnik
    dlzka_strany = obvod_kruznice / n
    nuholnik(n, dlzka_strany)

kruznica(50) #zavolame funkciu pre vykreslenie kruznice s polomerom 50 pixelov
```



Refaktorizácia

V predošlých príkladoch sme teda naprogramovali funkciu *nuholnik(n, dlzka_strany),* ktorá kreslí pravidelný mnohouholník. Následne sme pomocou nej naprogramovali funkciu *kruznica(polomer),* ktorá kreslí kružnicu.

Povedzme, že by sme teraz chceli definovať funkciu, ktorá bude kresliť **oblúk kružnice** zodpovedajúci nejakému *uhlu* a *polomeru* kružnice.

Problém je, že momentálne nedokážeme jednoducho vychádzať z funkcie pre kreslenie kružnice, pretože tá kreslí kružnicu podľa pravidelného nuholníka, t.j. útvaru, ktorý je uzatvorený.

Refaktorizácia

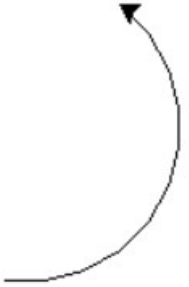
Naprogramujeme funkciu *nlomena(n, dlzka_strany, uhol)*, ktorá bude predstavovať lomenú čiaru pozostávajúcu z *n* úsečiek dĺžky *dlzka_strany* pričom korytnačka sa po každej čiare otočí vľavo o *uhol*. Pôjde o **zovšeobecnenie** funkcie *nuholnik(n, dlzka_strany)*. V *n*-uholníku totižto máme *n* strán, pričom korytnačka sa vždy otočí vľavo o uhol veľkosti 360/n, vďaka čomu je výsledný geometrický útvar uzatvorený.

Ak teraz nahradíme uhol uhlom o voliteľnej veľkosti, dosiahneme možnosť kresliť aj otvorené útvary, keďže výsledok bude **lomená čiara.**

Podstatou tohto “triku” je, že takáto čiara bude pre veľké *n a* vhodne zvolený uhol vyzerať ako kružnicový oblúk.

Refaktorizácia

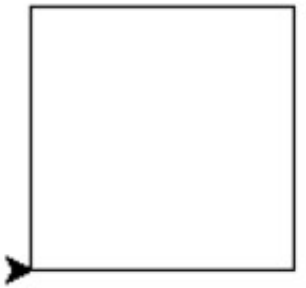
```
import turtle

def nlomena(n, dlzka_strany, uhol):
    for i in range(n):
        turtle.forward(dlzka_strany)
        turtle.left(uhol)

nlomena(10, 20, 15) #lomena ciara s 10 useckami dlzky 20 s uhlom 15
                    #medzi useckami
```



Refaktorizácia

Po naprogramovaní funkcie *nlomena(n, dlzka_strany, uhol)* môžeme následne **preprogramovať**, t.j. **zmeniť kód funkcie** *nuholnik(n, dlzka_strany)* a využiť v ňom volanie funkcie *nlomena* s uhlom veľkosti 360/n, keďže takto znovu nakreslíme pravidelný *n*-uholník!

```
import turtle

def nlomena(n, dlzka_strany, uhol):
    for i in range(n):
        turtle.forward(dlzka_strany)
        turtle.left(uhol)

def nuholnik(n, dlzka_strany):
    uhol = 360/n
    nlomena(n, dlzka_strany, uhol)

nuholnik(4, 100)
```



Refaktorizácia

Mali sme teda funkciu *nuholnik(n, dlzka_strany)*, ktorá fungovala korektne.

Avšak po naprogramovaní funkcie *nlomena()* sme sa vrátili do kódu funkcie *nuholnik()* a zmenili sme **telo funkcie** tak, aby zostala zachovaná funkcionalita (funkcia naďalej kreslí pravidelný *n*-uholník), avšak po novom využíva novú funkciu *nlomena()*.

Tento proces je príkladom tzv. **refaktorizácie (refactoring) kódu.**

Refaktorizácia kódu je zmena zmena zdrojového kódu, ktorá **zachováva funkcionalitu** (t.j. program robí to isté čo pred refaktorizáciou), avšak dôjde k zmene implementácie, napr. k využitiu nových funkcií alebo preprogramovaniu tela funkcie tak, aby bol nový zdrojový kód efektívnejší alebo čitateľnejší.

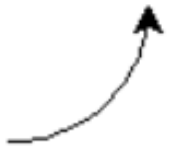
Refaktorizácia

Teraz pomocou funkcie *nlomena(n, dlzka_strany, uhol)* naprogramujeme funkciu kresliacu kružnicový oblúk ako lomenú čiaru, ktorej výsledná dĺžka bude predstavovať zhruba dĺžku požadovaného kružnicového oblúka, pričom lomená čiara bude pozostávať z dostatočného počtu úsečiek (napr. my zvolíme 30), aby sa opticky javila ako kružnicový oblúk.

Všimnite si, ako zabezpečíme, aby výsledný oblúk korešpondoval s požadovaným uhlom výseku z kružnice – ak požadovaný uhol výseku je *uhol* a lomená čiara bude pozostávať z *n* úsečiek , tak po každej úsečke v lomenej čiare sa korytnačka otočí vľavo o *uhol / n*.

Refaktorizácia

```
import turtle

def nlomena(n, dlzka_strany, uhol):
    for i in range(n):
        turtle.forward(dlzka_strany)
        turtle.left(uhol)

def nuholnik(n, dlzka_strany):
    uhol = 360/n
    nlomena(n, dlzka_strany, uhol)


def obluk(polomer, uhol):
    obvod_obluka = 2 * 3.1415 * polomer * (uhol/360)
    n = 30 #na aproximaciu obluku pouzijeme 30 dielikov v lomenej ciare
    dlzka_strany = obvod_obluka / n
    uhol_otocenia = uhol/n
    nlomena(n, dlzka_strany, uhol_otocenia)

obluk(50, 90)
```



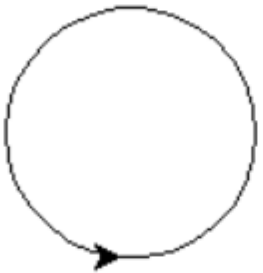
Refaktorizácia

Pomocou novej funkcie *obluk(polomer,uhol)* môžeme dokonca **refaktorizovať** funkciu *kruznica(polomer),* pretože kružnica nie je nič iné, len kružnicový oblúk s uhlom výseku 360 stupňov.

Preto zmeníme kód funkcie *kruznica(polomer)* tak, aby volal funkciu *obluk* s argumentom 360 pre parameter *uhol*.

Refaktorizácia

```
import turtle

def nlomena(n, dlzka_strany, uhol):
    for i in range(n):
        turtle.forward(dlzka_strany)
        turtle.left(uhol)

def nuholnik(n, dlzka_strany):
    uhol = 360/n
    nlomena(n, dlzka_strany, uhol)


def obluk(polomer, uhol):
    obvod_obluka = 2 * 3.1415 * polomer * (uhol/360)
    n = 30 #na aproximaciu obluku pouzijeme 30 dielikov v lomenej ciare
    dlzka_strany = obvod_obluka / n
    uhol_otocenia = uhol/n
    nlomena(n, dlzka_strany, uhol_otocenia)

def kruznica(polomer):
    obluk(polomer, 360)

kruznica(50) #zavolame funkciu pre vykreslenie kruznice s polomerom 50 pixelov
```



Rozhranie

Návrh funkcie pozostáva z 2 častí:
1) **Rozhranie (interface)** funkcie
2) Samotná implementácia funkcie

Rozhranie (interface) funkcie popisuje:
a) ako sa funkcia používa / čo funkcia robí
b) aký je identifikátor funkcie (jej meno)
c) aké má vstupné parametre
d) aké sú jej výstupy

Platí teda, že rozhranie funkcie je dôležité pre písanie programu, v ktorom funkciu používame – často nám pri používaní funkcií stačí vedieť čo robia, aké parametre majú a aké hodnoty vracajú, nemusí nás zaujímať, AKO je funkcia implementovaná!

Rozhranie

Videli sme 2 rôzne implementácie funkcie *kruznica(polomer):*

```
def kruznica(polomer):
    obvod_kruznice = 2 * 3.1415 * polomer
    n = 30 #na aproximaciu kruznice pouzijeme 30-uholnik
    dlzka_strany = obvod_kruznice / n
    nuholnik(n, dlzka_strany)
```

Versus

```
def kruznica(polomer):
    obluk(polomer, 360)
```

Obe verzie funkcie majú **rovnaké rozhranie** – kreslia kružnicu, nič nevracajú, majú 1 vstupný parameter – polomer kružnice - avšak majú rôzne implementácie.

Dokumentačné reťazce (docstrings)

Dôležitou súčasťou každého zdrojového kódu sú komentáre – o tom, ako vytvárať komentáre v Python-e sme si hovorili na prvej prednáške.

Jedným zo základných typov komentárov sú tzv. dokumentačné reťazce, angl. docstrings.

Dokumentačný reťazec je krátky komentár uvedený pri hlavičke funkcie, ktorý popisuje rozhranie (interface) funkcie:

a) Popisuje, čo funkcia robí **bez toho**, že by zachádzal do implementačných detailov.
b) Popisuje, čo predstavujú vstupné parametre.
c) Ak to nie je očividné, popisuje akého typu sú jednotlivé parametre.
d) Popisuje, čo funkcia vracia.

Dokumentačné reťazce (docstrings)

```
def nlomena(n, dlzka_strany, uhol):
    """
    Kresli lomenu ciaru pozostavajucu z useciek rovnakej dlzky
    a rovnakeho uhla medzi useckami.

    n: celociselny pocet useciek
    dlzka_strany: dlzka jednej usecky
    uhol: uhol medzi useckami (v stupnoch)
    """
    for i in range(n):
        turtle.forward(dlzka_strany)
        turtle.left(uhol)
```

Príklad

1) Napíšte program, ktorý sčíta čísla od 1 po 10 a súčet vypíše na obrazovku.

2) Program z úlohy 1 **zapúzdrite** do funkcie, ktorá vypočítaný súčet vráti.

3) **Zovšeobecnite** funkciu z úlohy 2, aby vrátila súčet čísiel od 1 po n.

4) **Zovšeobecnite** funkciu z úlohy 3, aby vrátila súčet k-tych mocnín čísiel od 1 po n.

Úloha 1)

1) Napíšte program, ktorý sčíta čísla od 1 po 10 a súčet vypíše na obrazovku.

```
sucet = 0
for i in range(1, 10+1):
    sucet = sucet + i

print(sucet)
```

Úloha 2)

2) Program z úlohy 1 **zapúzdrite** do funkcie, ktorá vypočítaný súčet vráti.

```
def suma():
    sucet = 0
    for i in range(1, 10+1):
        sucet = sucet + i
    return sucet
```

Úloha 3)

3) **Zovšeobecnite** funkciu z úlohy 2, aby vrátila súčet čísiel od 1 po n.

```
def suma(n):
    sucet = 0
    for i in range(1, n+1):
        sucet = sucet + i
    return sucet
```

Všimnite si, že pri zovšeobecnení sme do funkcie doplnili parameter *n*.
Zároveň pre voľbu parametra *n = 10* sa program správa ako predošlá verzia (úloha 2) pred zovšeobecnením.

Úloha 4)

4) **Zovšeobecnite** funkciu z úlohy 3, aby vrátila súčet k-tych mocnín čísiel od 1 po n.

```
def suma(n,k):
    sucet = 0
    for i in range(1, n+1):
        sucet = sucet + i**k
    return sucet
```

Všimnite si, že pri zovšeobecnení sme do funkcie doplnili parameter *k*.
Zároveň pre voľbu parametra k = *1* sa program správa ako predošlá verzia (úloha 3) pred zovšeobecnením.

Voliteľné parametre

- V Pythone je možné nastaviť niektoré parametre vo funkcii ako voliteľné, t.j. pri volaní funkcie nie je nutné odovzdať do nich hodnoty.
- To sa dosiahne tak, že tieto parametre majú nejaké **predvolené** hodnoty a ak sa im nedodá hodnota pri volaní, tak funkcia použije predvolenú hodnotu.
- Predvolené hodnoty parametrov sa špecifikujú v hlavičke funkcie tak, že sa priamo pri parametri uvedie hodnota, ktorou sa inicializuje, t.j.:

def *meno_funkcie*(*parameter* = predvolena_hodnota,...)

Voliteľné parametre

Ak by sme napríklad vo funkcii *suma(n,k)* chceli stanoviť, že *k* je voliteľný parameter a v prípade, že preň používateľ neuvedie hodnotu pri volaní funkcie, tak sa ako základná hodnota, ktorou sa *k* inicializuje, vezme *k = 1*, takú funkciu by sme definovali ako:

test.py - Z:\Documents\prednaska\test.py (3.12.6)

File Edit Format Run Options Window Help

```
def suma(n,k = 1):
    sucet = 0
    for i in range(1, n+1):
        sucet = sucet + i**k
    return sucet

print(suma(10,1)) #volanie s explicitne uvedenymi obomi argumentami, n = 10, k = 1
print(suma(10,2)) #volanie s explicitne uvedenymi obomi argumentami, n = 10, k = 2
print(suma(10))   #volanie s explicitne uvedenym n = 10, k je volitelny parameter
                  #a teda ked sa neuvedie hodnota pre k, tak sa nastavi na 1
```

Ln

```
55
385
55
```