**4 - Podmienený cyklus**

V Pythone existuje konštrukcia cyklu, ktorá opakuje vykonávanie postupnosti príkazov v závislosti od nejakej podmienky:

while podmienka: # opakuj príkazy, kým platí podmienka

prikaz

prikaz

...

Vidíme podobnosť s podmieneným príkazom if - vetvením. Tento nový príkaz postupne:

* zistí hodnotu podmienky, ktorá je zapísaná za slovom while
* ak má táto podmienka hodnotu False, blok príkazov, ktorý je telom cyklu, sa preskočí a pokračuje sa na nasledovnom príkaze za celým while-cyklom (podobne ako v príkaze if bez vetvy else), hovoríme, že sa ukončilo vykonávanie cyklu
* ak má podmienka hodnotu True, **vykonajú sa všetky príkazy** v tele cyklu (v odsunutom bloku príkazov)
* a znovu sa testuje podmienka za slovom while, t.j. celé sa to opakuje

Najprv zapíšeme pomocou tohto cyklu, to čo už vieme pomocou for-cyklu:

for i in range(1, 21):

print(i, i\*i)

Vypíše tabuľku druhých mocnín čísel od 1 do 20. Prepis na cyklus while znamená, že zostavíme podmienku, ktorá bude testovať, napríklad premennú i: tá nesmie byť väčšia ako 20. Samozrejme, že už pred prvou kontrolou premennej i v podmienke cyklu while, musí mať nejakú hodnotu:

i = 1

while i < 21:

print(i, i\*i)

i += 1

V cykle sa vykoná print() a zvýši sa hodnota premennej i o jedna.

while-cykly sa ale častejšie používajú vtedy, keď zápis pomocou for-cyklu je príliš komplikovaný, alebo sa ani urobiť nedá.

Ukážeme to na programe, ktorý bude do jedného radu tesne vedľa seba kresliť stále sa zväčšujúce štvorce postupne so stranami 10, 20, 30, … Pritom bude dávať pozor, aby naposledy nakreslený štvorec „nevypadol“ z plochy - teda chceme skončiť skôr, ako by sme nakreslili štvorec, ktorý sa už celý nezmestí do grafickej plochy. Štvorce so stranou a budeme kresliť takto:

canvas.create\_rectangle(x, 200, x+a, 200-a)

vďaka čomu, všetky ležia na jednej priamke (y = 200). Keď teraz budeme posúvať x-ovú súradnicu vždy o veľkosť nakresleného štvorca, ďalší bude ležať tesne vedľa neho.

Program pomocou while-cyklu zapíšeme takto:

import tkinter

sirka = int(input('šírka plochy: '))

canvas = tkinter.Canvas(width=sirka)

canvas.pack()

x = 5

a = 10

while x + a < sirka:

canvas.create\_rectangle(x, 200, x+a, 200-a, fill='white')

x += a

a += 10

# príkazy za cyklom

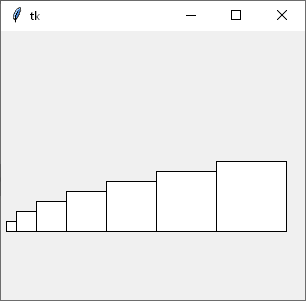
Program pracuje korektne pre rôzne šírky grafickej plochy. Ak zväčšovanie strany štvorca a += 10 nahradíme a = 2 \* a, program bude pracovať aj s takto zväčšovanými štvorcami (strany budú postupne 10, 20, 40, 80, …).

Zhrňme, ako funguje tento typ cyklu:

1. vyhodnotí sa podmienka x + a < sirka, t.j. pravý okraj štvorca, ktorý práve chceme nakresliť, sa ešte celý zmestí do grafickej plochy
2. ak je podmienka splnená (pravdivá), postupne sa vykonajú všetky príkazy, t.j. nakreslí sa ďalší štvorec so stranou a a potom sa posunie x (ľavý okraj budúceho štvorca) o veľkosť práve nakresleného štvorca a a tiež sa ešte zmení veľkosť budúceho štvorca a o 10
3. po vykonaní tela cyklu sa pokračuje v 1. kroku, t.j. opäť sa vyhodnotí podmienka
4. ak podmienka nie je splnená (nepravda), cyklus končí a ďalej sa pokračuje v príkazoch za cyklom

Uvedomte si, že podmienka nehovorí, kedy má cyklus skončiť, ale naopak - kým podmienka platí, vykonávajú sa všetky príkazy v tele cyklu.

Pre šírku grafickej plochy 300 dostávame takýto obrázok:



Konkrétne tento program s while-cyklom vieme jednoducho prepísať pomocou for-cyklu a break:

import tkinter

sirka = int(input('šírka plochy: '))

canvas = tkinter.Canvas(width=sirka)

canvas.pack()

x = 5

for a in range(10, sirka, 10):

if x + a >= sirka:

break

canvas.create\_rectangle(x, 200, x+a, 200-a, fill='white')

x += a

Vyššie sme zostavili program, ktorý zisťoval, či je zadané číslo prvočíslo. Použili sme for-cyklus, v ktorom sme zadané číslo postupne delili všetkými číslami, ktoré sú menšie ako samotné číslo. Zistili sme, že na zisťovanie prvočísla nepotrebujeme skutočný počet deliteľov, ale malo by nám stačiť zistenie, či existuje aspoň jeden deliteľ. Keď sa vyskytne prvý deliteľ (t.j. platí cislo % delitel != 0), cyklus môžeme ukončiť a vyhlásiť, že číslo nie je prvočíslo. Ak ani jedno číslo nie je deliteľom nášho čísla, hodnota premennej delitel dosiahne cislo a to je situácia, keď cyklus tiež skončí (t.j. keď delitel == cislo, našli sme prvočíslo). Zapíšeme to while-cyklom:

cislo = int(input('Zadaj číslo: '))

delitel = 2

while delitel < cislo and cislo % delitel != 0:

delitel = delitel + 1

if delitel == cislo:

print(cislo, 'je prvočíslo')

else:

print(cislo, 'nie je prvočíslo')

Do podmienky while-cyklu sme pridali novú časť. Operátor and tu má ten význam, že na to, aby sa cyklus opakoval, musia byť splnené obe časti. Uvedomte si, že **cyklus skončí vtedy**, keď prestane platiť zadaná podmienka, t.j. negácia podmienky (matematicky to upravíme):

* not (delitel < cislo and cislo % delitel != 0)
* not delitel < cislo or not cislo % delitel != 0
* delitel >= cislo or cislo % delitel == 0

while-cyklus teda skončí vtedy, keď delitel >= cislo, **alebo** cislo % delitel == 0 (teda, že deliteľ by bol už zbytočne veľký alebo našli sme hodnotu, ktorá delí naše číslo).

**Zisťovanie druhej odmocniny**

Ukážeme, ako zistíme druhú odmocninu čísla aj bez volania funkcie math.sqrt(x), resp. umocňovaním na jednu polovicu x\*\*0.5.

Prvé riešenie:

cislo = float(input('zadaj číslo:'))

x = 1

while x\*\*2 < cislo:

x += 1

print('odmocnina', cislo, 'je', x)

Takto nájdené riešenie môže byť dosť nepresné, lebo x zvyšujeme o 1, takže, napríklad odmocninu z 26 vypočíta ako 6. Skúsme zjemniť krok, o ktorý sa mení hľadané x:

cislo = float(input('zadaj číslo:'))

x = 1

while x\*\*2 < cislo:

x += 0.001

print('odmocnina', cislo, 'je', x)

Teraz dáva program lepšie výsledky, ale pre väčšiu zadanú hodnotu mu to trvá citeľne dlhšie - skúste zadať, napríklad 10000000. Keďže mu vyšiel výsledok približne 3162.278 a dopracoval sa k nemu postupným pripočítavaním čísla 0.001 k štartovému 1, musel urobiť vyše 3 miliónov pripočítaní a tiež toľkokrát testov vo while-cykle (podmienky x\*\*2 < cislo). Pre toto je takýto algoritmus nepoužiteľne pomalý.

Využijeme inú ideu:

* zvolíme si interval, v ktorom sa určite bude nachádzať hľadaný výsledok (hľadaná odmocnina), napríklad nech je to interval <1, cislo> (pre čísla väčšie ako 1 je aj odmocnina väčšia ako 1 a určite je menšia ako samotne cislo)
* ako x (prvý odhad našej hľadanej odmocniny) zvolíme stred tohto intervalu
* zistíme, či je druhá mocnina tohto x väčšia ako zadané cislo alebo menšia
* ak je väčšia (x je už zbytočne veľké), tak upravíme predpokladaný interval, tak že jeho hornú hranicu zmeníme na x
* ak je ale menšia, upravíme dolnú hranicu intervalu na x
* tým sa nám interval zmenšil na polovicu
* toto celé opakujeme, kým už nie je nájdené x dostatočne blízko k hľadanému výsledku, t.j. či sa nelíši od výsledku menej ako zvolený rozdiel (epsilon)

Zapíšme to:

cislo = float(input('zadaj číslo:'))

od = 1

do = cislo

x = (od + do) / 2

pocet = 0

while abs(x\*\*2 - cislo) > 0.001:

if x\*\*2 > cislo:

do = x

else:

od = x

x = (od + do) / 2

pocet += 1

print('druhá odmocnina', cislo, 'je', x)

print('počet prechodov while-cyklom bol', pocet)

Ak spustíme program pre 10000000 dostávame:

zadaj číslo:10000000

druhá odmocnina 10000000.0 je 3162.2776600480274

počet prechodov while-cyklom bol 44

čo je výrazné zlepšenie oproti predchádzajúcemu riešeniu, keď prechodov while-cyklom (hoci jednoduchších) bolo vyše 3 milióny.

**Nekonečný cyklus**

Cyklus s podmienkou, ktorá má stále hodnotu True, bude nekonečný. Napríklad:

i = 0

while i < 10:

i -= 1

Nikdy neskončí, lebo premenná i bude stále menšia ako 10. Takéto výpočty môžeme prerušiť stlačením klávesov **Ctrl/C**.

Aj nasledovný cyklus je úmyselne nekonečný:

while 1:

pass

Pripomíname, že príkaz pass je **prázdny príkaz**, ktorý nerobí nič. V tomto príklade pass označuje prázdne telo cyklu a teda tento cyklus bude nikdy nekončiaci.

Už vieme, že príkaz break môžeme použiť v tele for-cyklu a vtedy sa zvyšok cyklu nevykoná. Nasledovný príklad ilustruje použitie break aj vo while-cykle:

sucet = 0

while True:

retazec = input('zadaj číslo: ')

if retazec == '':

break

sucet += int(retazec)

print('súčet prečítaných čísel =', sucet)

V tomto príklade sa čítajú čísla zo vstupu, kým nezadáme prázdny reťazec: vtedy cyklus končí a program vypíše súčet prečítaných čísel. Napríklad:

zadaj číslo: 5

zadaj číslo: 17

zadaj číslo: 2

zadaj číslo:

súčet prečítaných čísel = 24