Oznam

Budúci týždeň vo štvrtok 17.10. je študijné voľno.
V stredu 16.10. sa učí **podľa štvrtkového rozvrhu.**

Teda v stredu 16.10. budú cvičenia v C117 v časoch od 10:00, od 13:00, od 15:00, od 17:00 pre „štvrtkových“ študentov. „Stredajší“ študenti sa môžu v stredu zúčastniť ľubovoľného cvičenia, ak im to ich rozvrh dovolí.

Pre „stredajších“ študentov, ktorí mali mať cvičenie, ale nie je účasť na cvičeniach budúci týždeň povinná.

Odporúčam priniesť si vlastné počítače (ak môžete).

PROG1: Prednáška 4

Podmienené príkazy

Vstupy z klávesnice

- Doteraz sme vždy mali situácie, že hodnoty boli v programe niekde už zadané.

- Avšak do programu môžeme hodnoty vkladať aj počas jeho behu, napríklad z klávesnice. To znamená, program môže vyzvať používateľa, aby zadal nejakú hodnotu z klávesnice (číslo, reťazec, …) a program hodnotu ďalej spracuje.

- Vďaka tomu možno vytvárať **interaktívne programy**, t.j. programy, s ktorými môže používateľ interagovať. Programy potom môžu alebo tento používateľov vstup ďalej spracovať, alebo vykonávať rôzne činnosti v závislosti od tohto používateľského vstupu.

Vstupy z klávesnice

- Načítanie vstupu od používateľa sa robí v jazyku Python pomocou funkcie **input**().

- Keď sa zavolá funkcia **input()** v jazyku Python, program sa pozastaví a počká, kým používateľ zadá z klávesnice nejaký vstup a stlačí Enter. Po stlačení Enter program pokračuje v činnosti a funkcia **input() vráti** to, čo používateľ zadal, ako svoju návratovú hodnotu.

- Funkcia **input() VŽDY** vracia návratovú hodnotu ako typ **string**, t.j. **VŽDY** vráti presne to, čo používateľ zadal na klávesnici ako postupnosť príslušných textových znakov.

Vstupy z klávesnice

prednaska4.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/predr

File Edit Format Run Options Window Help

```
vstup1 = input()
vstup2 = input()
vstup3 = vstup1+vstup2
print(vstup3)
```

Do premennej vstup3 sa vloží "+" premenných vstup1 a vstup2. Keďže obe premenné vstup1 a vstup2 obsahujú reťazce, v jazyku Python operátor "+" reťazcov znamená ich zreťazenie.

```
ahoj
student
ahojstudent
>>>
```

Tu používateľ zadal *ahoj* ako prvý vstup z klávesnice, ten sa uložil do premennej vstup1 ako reťazec

Tu používateľ zadal *student* ako druhý vstup z klávesnice, ten sa uložil do premennej vstup2 ako reťazec

Keďže zreťazenie reťazcov *ahoj* a *student* je reťazec *ahojstudent* pri výpise vstup3 na obrazovku sa vypísalo *ahojstudent*

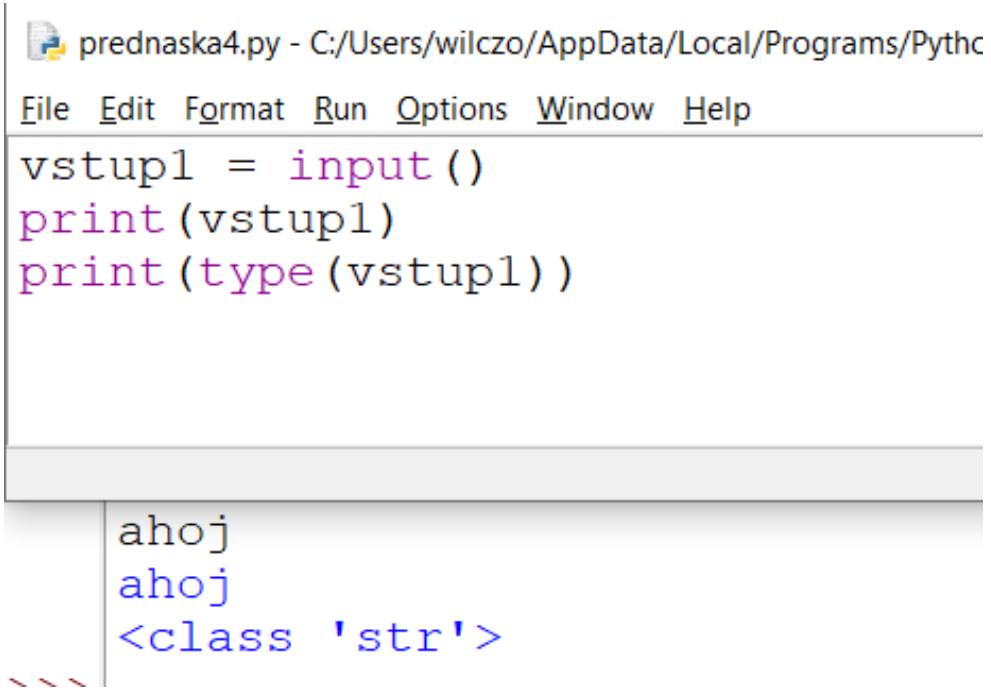
Vstupy z klávesnice

O tom, že funkcia **input()** vždy vracia načítaný vstup z klávesnice ako reťazec sa vieme presvedčiť aj tak, že použijeme funkciu **type(),** do ktorej ako argument vložíme nejakú hodnotu/premennú.

Funkcia **type()** je vstavaná funkcia v jazyku Python, ktorá vráti **typ hodnoty**, ktorú dostala ako svoj vstupný argument.

Ak teda do funkcie **type()** vložíme reťazec (hodnotu typu **string)** alebo **premennú**, kde sa reťazec nachádza, funkcia **type()** vráti hodnotu <class ‘str‘>.
(Teraz nie je dôležité, čo táto hodnota predstavuje, dôležitá je časť ‘str‘ ktorá hovorí o tom, že ide o **string**, teda reťazec.

Vstupy z klávesnice



Vstupy z klávesnice

Čo vypíše nasledovný program, ak ako prvý vstup zadáme *1* a ako druhý vstup *2* ?

```
vstup1 = input()
vstup2 = input()
vstup3 = vstup1 + vstup2
print(vstup3)
```

Program vypíše 12, pretože obe hodnoty sú **reťazce** a „+“ medzi reťazcami robí **zreťazenie!**

```
1
2
12
```

Vstupy z klávesnice

Preto platí, že ak načítame z klávesnice čísla a chceme, aby ich Python interpretoval ako **číselné hodnoty,** musíme tieto hodnoty **pretypovať!!!**

Pretypovanie hodnoty na **celočíselnú hodnotu** urobíme pomocou volania funkcie **int(*hodnota).***
Funkcia **int(*hodnota*) vráti** celočíselnú verziu svojho vstupného argumentu.

Podobne funkcia **float(*hodnota)* vráti** desatinnú verziu svojho vstupného argumentu.

Často sa pretypovanie robí priamo pri načítaní vstupu, t.j. ak chcem načítanú hodnotu interpretovať ako celé číslo, príslušné volanie bude **int(input())**

Ak chcem načítanú hodnotu interpretovať ako desatinné (float) číslo, volanie bude **float(input())**

Vstupy z klávesnice

prednaska4.py - C:/Users/wilczo/AppData/Local/Programs/Py

File Edit Format Run Options Window Help

```
vstup1 = int(input())
vstup2 = int(input())
vstup3 = vstup1 + vstup2
print(vstup3)
```

```
1
2
3
```

<--- Tu je vidieť ako sa bude program správať, ked oba vstupy hneď pri načítaní pretypujeme na celé číslo (int). Uvedomte si, že volanie int(input()) funguje tak, že ako vstup do int() vstupuje návratová hodnota input(), t.j. vstup od používateľa.
Ako vidíme, v premenej *vstup3* je hodnota 3 (pretože obe hodnoty 1 a 2 chápe program ako čísla).

---

prednaska4.py - C:/Users/wilczo/AppData/Local/Pr

File Edit Format Run Options Window Help

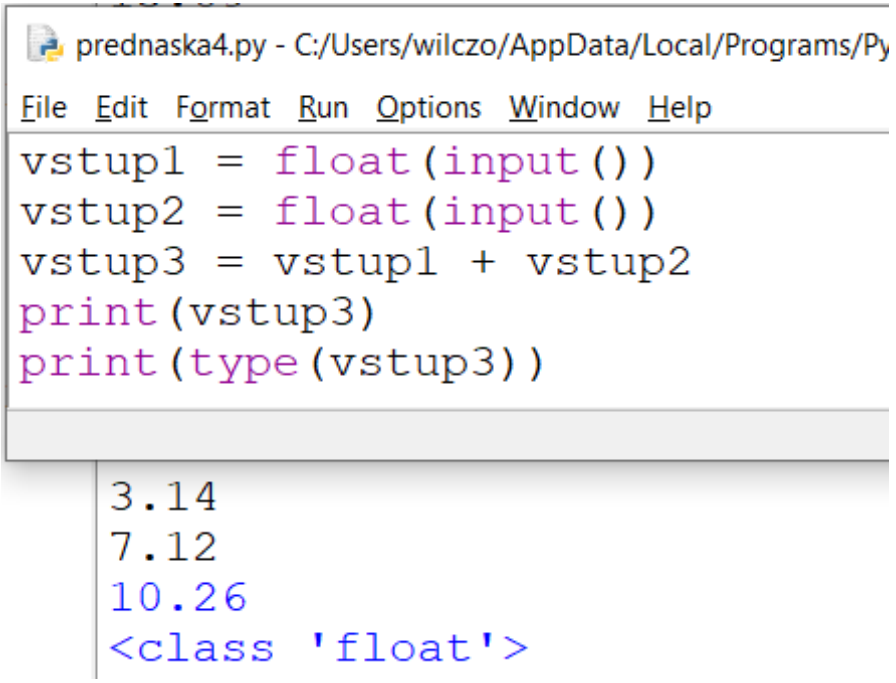
```
vstup1 = int(input())
vstup2 = int(input())
vstup3 = vstup1 + vstup2
print(type(vstup1))
print(type(vstup2))
print(type(vstup3))
```

```
1
2
<class 'int'>
<class 'int'>
<class 'int'>
```

<--- Tu je vidieť, čo sa stane, keď si dáme vypísať typ premenných vstup1, vstup2, vstup3. Keďže vo všetkých sú celé čísla, program vypíše <class ‘int’>.
Znovu, nie je úplne dôležité, čo presne to znamená (class sú „triedy“, to je niečo, čo sa budete učiť na predmete B-OOP), dôležité je slovo ‘int‘, teda že ide o integer (celé číslo).

Vstupy z klávesnice

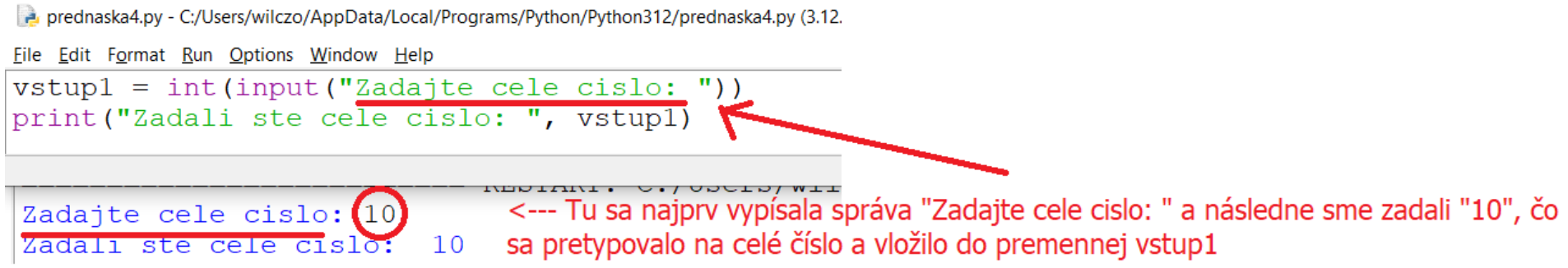
Analogicky pre desatinné čísla (float)



Vstupy z klávesnice

Funkcia **input()** môže mať aj vstupný argument!

V tomto prípade je to textový reťazec, ktorý sa vypíše na výstup v momente, keď bude program očakávať vstup od používateľa! Spravidla je to vhodné použiť na výpis nejakej správy pre používateľa, aká hodnota sa od neho očakáva!



Vstupy z klávesnice

prednaska4.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska4.py (3.12.6)

File Edit Format Run Options Window Help

```
def parne(n):
    """
    Funkcia vypise prvych n kladnych parnych cisiel.

    n: cele cislo udavajuce pocet parnych cisiel
    """
    for i in range(n):
        print((i+1)*2)

def parne_interaktivna_verzia():
    vstup = int(input("Zadajte kolko parnych cisiel chcete vypisat: "))
    parne(vstup)

parne_interaktivna_verzia()
```

```
Zadajte kolko parnych cisiel chcete vypisat: 5
2
4
6
8
10
```

Operátor zvyšku po delení

Pre pripomenutie z prvej prednášky:
- v programovaní často využívame matematické operácie, jednou z typických je výpočet zvyšku po delení

- v Pythone existuje operátor % (nazývaný aj modulo), pomocou ktorého vieme počítať zvyšok po delení.

Výraz:

cislo1 % cislo2

má hodnotu zvyšku z čísla *cislo1* po delení číslom *cislo2*

Operátor % (modulo)

prednaska4.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska4.py (3.12.6)

File Edit Format Run Options Window Help

```
cislo1 = 20
for i in range(1, cislo1+1):
    print("Zvysok ",cislo1," po deleni cislom ",i," = ",cislo1%i)
```

```
Zvysok  20  po deleni cislom  1  =  0
Zvysok  20  po deleni cislom  2  =  0
Zvysok  20  po deleni cislom  3  =  2
Zvysok  20  po deleni cislom  4  =  0
Zvysok  20  po deleni cislom  5  =  0
Zvysok  20  po deleni cislom  6  =  2
Zvysok  20  po deleni cislom  7  =  6
Zvysok  20  po deleni cislom  8  =  4
Zvysok  20  po deleni cislom  9  =  2
Zvysok  20  po deleni cislom  10  =  0
Zvysok  20  po deleni cislom  11  =  9
Zvysok  20  po deleni cislom  12  =  8
Zvysok  20  po deleni cislom  13  =  7
Zvysok  20  po deleni cislom  14  =  6
Zvysok  20  po deleni cislom  15  =  5
Zvysok  20  po deleni cislom  16  =  4
Zvysok  20  po deleni cislom  17  =  3
Zvysok  20  po deleni cislom  18  =  2
Zvysok  20  po deleni cislom  19  =  1
Zvysok  20  po deleni cislom  20  =  0
```

Malá matematická poznámka:

Všimnite si, že v prípade, že hodnota *i* delí číslo 20, tak je zvyšok po delení 0.

(to je samozrejme logické, len som na to chcel upozorniť)

Booleovské hodnoty a booleovské výrazy

Okrem typov hodnôt, ktoré sa v programovaní používajú a ktoré sme si spomínali, ako:
- celé čísla
- desatinné čísla
- reťazce

sa v programovaní často používa aj špeciálny typ hodnôt, tzv. Booleovské hodnoty: True/False.
Alternatívne by sme ich vedeli označiť slovenským termínom Pravda/Nepravda, resp. True/False, prípadne v dvojkovej logike 1/0.

Výrazy nadobúdajúce hodnoty True/False nazývame **booleovské výrazy**.

Booleovské výrazy

Aj jazyk Python poskytuje možnosť využívať booleovské hodnoty: **True** a **False**

Tieto hodnoty predstavujú Pythonovskú reprezentáciu hodnôt Pravda / Nepravda

Booleovský výraz v jazyku Python je teda výraz, ktorý nadobúda hodnoty **True** alebo **False**

Booleovské výrazy

S hodnotami True/False sa v jazyku Python vieme stretnúť napríklad vtedy, ak použijeme operátor == , čo je operátor, pomocou ktorého vieme porovnať 2 hodnoty a otestovať, či sú rovnaké alebo nie.

Napríklad, ak by sme mali premenné *a,b* potom výraz: (*a* == *b)*
by nadobudol hodnoty True alebo False podľa toho, či Python považuje hodnotu v premennej *a* za rovnakú hodnotu ako v premennej *b* alebo nie.

V tomto prípade by teda (*a*==*b*) bol príklad booleovského výrazu.

Výraz, v ktorom vystupuje operátor == teda nadobudne hodnotu True, ak sa ľavý operand rovná pravému (t.j. hodnota vľavo od == sa rovná hodnote vpravo od ==) a hodnotu False, ak sa ľavý operand nerovná pravému operandu.

Booleovské hodnoty

prednaska4.py - C:/Users/wilczo/AppData/Local/Programs/Python/Pytl

File Edit Format Run Options Window Help

```
print(5 == 5)
print(5 == 6)
a = 1
b = 2
vysledok_porovnania = (a == b)
print(vysledok_porovnania)
print(type(vysledok_porovnania))
```

```
True
False
False
<class 'bool'>
```

Všimnite si, že hodnotu výrazu *a==b* (teda v TOMTO prípade hodnotu False, pretože hodnoty *a* a *b* sa nerovnajú) je možné vložiť do inej premennej.

V premennej "vysledok_porovnania" je teda hodnota False

Hodnoty True alebo False sú hodnoty typu 'bool' (špeciálny booleovský typ v jazyku Python)

Booleovské hodnoty - komparátory

Operátory, ktoré vedia porovnať 2 hodnoty a vrátiť hodnotu True / False podľa toho, aký je výsledok porovnania, sa nazývajú **komparátory** (angl. compare = porovnať), prípadne „relačné operátory“. V jazyku Python sa používajú tieto:

== (a == b) znamená „*a* je rovnaké ako *b*“
!= (*a* != *b)* znamená „*a* nie je rovnaké ako *b*“
> (a > b) znamená „*a* je väčšie ako *b*“
< (*a* < *b)* znamená „*a* je menšie ako *b*“
>= (a >= b) znamená „*a* je väčšie alebo rovné ako *b*“
<= (*a* <= *b)* znamená „*a* je menšie alebo rovné ako *b*“

POZOR! Porovnanie rovnosti sa robí pomocou dvoch symbolov „==“
Jedno „=“ sa používa ako operátor priradenia, teda:

a = *b* priradí hodnotu premennej *b* do premennej *a*
a == *b* nadobudne hodnotu True, ak je *a* rovnaké ako *b*, inak nadobudne hodnotu False

Booleovské hodnoty - komparátory

prednaska4.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska4.py (3.12.6)

File Edit Format Run Options Window Help

```
a = 10
b = 20
print(a==b) #False, pretoze otazka ci sa 10 rovna 20 ma odpoved NIE
print(a!=b) #True, pretoze otazka ci sa 10 nerovna 20 ma odpoved ANO
print(a>b)  #False, pretoze otazka ci je 10 vacsie ako 20 ma odpoved NIE
print(a<b)  #True, pretoze otazka ci je 10 mensie ako 20 ma odpoved ANO
print(a>=b) #False, pretoze otazka ci je 10 vacsie alebo rovne ako 20 ma odpoved NIE
print(a<=b) #True, pretoze otazka ci je 10 mensie alebo rovne ako 20 ma odpoved ANO
```

Ln: 8

```
False
True
False
True
False
True
```

Logické operátory

S booleovskými hodnotami *True* a *False*, respektíve s booleovskými výrazmi, ktoré môžu nadobúdať hodnoty *True* a *False*, môžeme vykonávať 3 typy logických operácií pomocou 3 logických operátorov:

1) operátor **and**
- logický operátor **and** predstavuje logický súčin (konjunkciu) 2 booleovských hodnôt

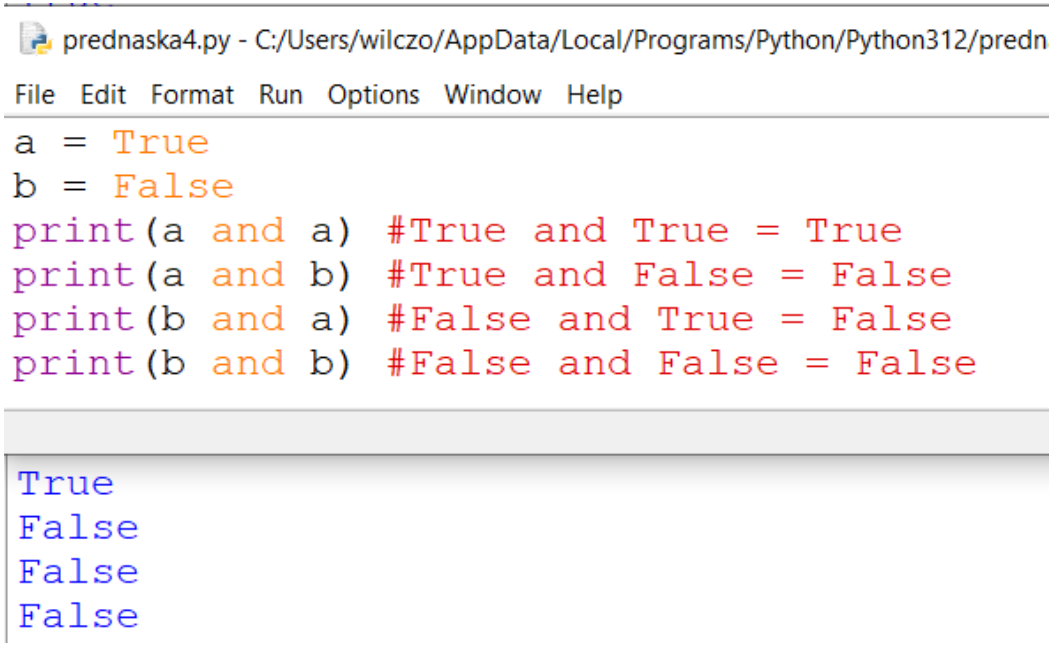
2) operátor **or**
- logický operátor **or** predstavuje logický súčet (disjunkciu) 2 booleovských hodnôt

3) operátor **not**
- logický operátor **or** predstavuje negáciu 1 booleovskej hodnoty

V jazyku Python sú tieto operátory nazvané rovnako, t.j. **and, or, not.**

Logický operátor and

Logický operátor **and** vráti hodnotu **True** ak oba operandy majú hodnotu **True**, t.j. ak robíme operáciu **and** medzi 2 hodnotami, ktoré majú hodnotu **True.**



Logický operátor or

Logický operátor **or** vráti hodnotu **True** ak aspoň 1 operand má hodnotu **True**, t.j. ak robíme operáciu **or** medzi 2 hodnotami, kde aspoň 1 má hodnotu **True.**

prednaska4.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednas

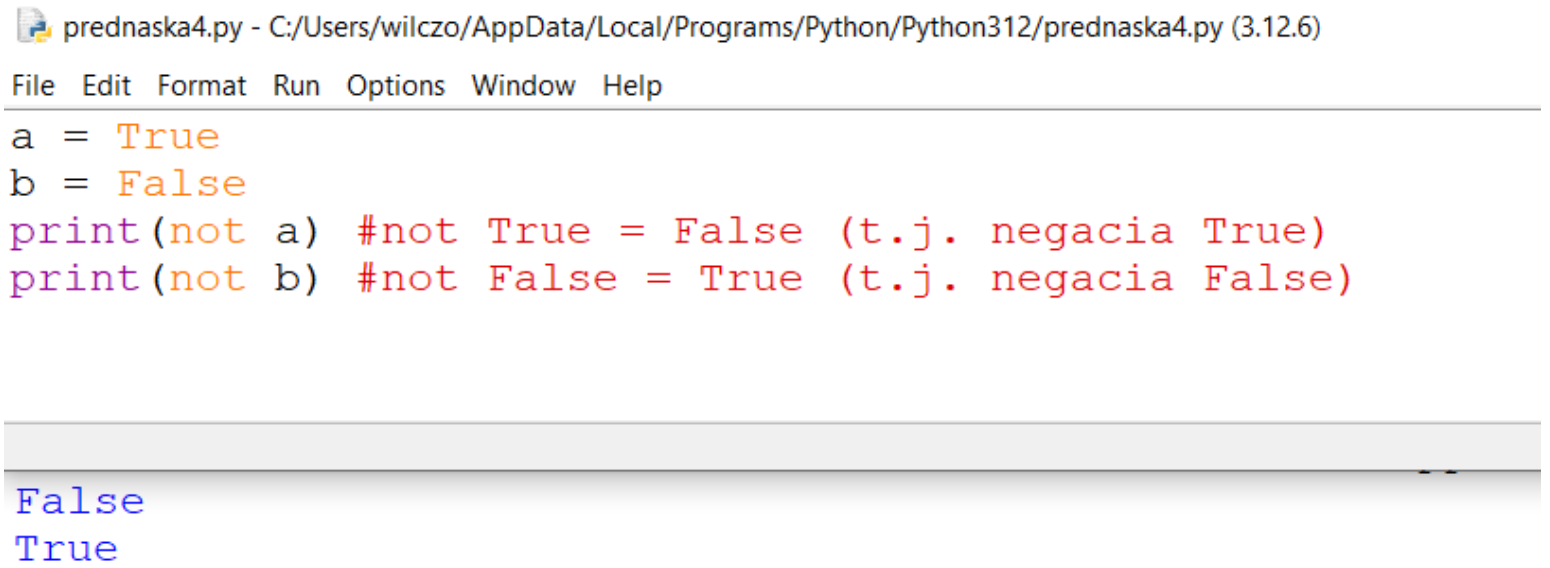
File Edit Format Run Options Window Help

```
a = True
b = False
print(a or a) #True or True = True
print(a or b) #True or False = True
print(b or a) #False or True = True
print(b or b) #False or False = False
```

```
True
True
True
False
```

Logický operátor not

Logický operátor **not** vráti hodnotu **True** ak operand má hodnotu **False** a naopak, operátor **not** vráti hodnotu **False** ak operand má hodnotu **True.**



Ďalšie príklady

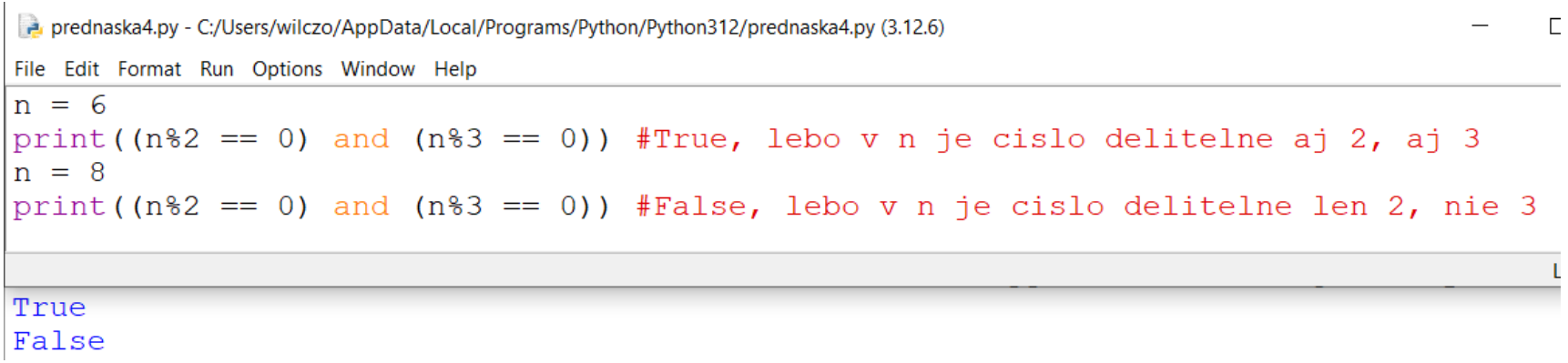
prednaska4.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska4.py (3.12.6)

File Edit Format Run Options Window Help

```
print((1 < 2) and (2 < 3)) #True lebo oba vyrazy (1<2) a (2<3) su True
print((1 < 2) and (2 > 3)) #False, lebo druhy vyraz (2>3) je False
print((1 < 2) or (2 > 3)) #True, lebo aspon jeden vyraz, (1<2), je True
print((1 > 2) or (2 > 3)) #False, lebo oba vyrazy (1>2),(2>3) su False
print(not (1<2))          #False, lebo (1<2) je True a jeho negaciou je False
print(not (1>2))          #True, lebo (1>2) je False a jeho negaciou je True
```

```
True
False
True
False
False
True
```

Ďalšie príklady



Pomocou logických operátorov a logických výrazov teda vieme zostrojiť aj zložitejšie logické výrazy, ako napríklad vyššie uvedený zložený logický výraz

(n%2 == 0) and (n%3 == 0)

ktorý nadobúda hodnotu True / False podľa toho, či je *n* deliteľné 2 a zároveň 3.

Podmienené príkazy

Základne využitie booleovských hodnôt, booleovských výrazov a logických operátorov pri programovaní a navrhovaní algoritmov je v tzv. **podmienených príkazoch.**

Podmienené príkazy

Doteraz sme mali programy, ktoré obsahovali prácu s premennými, matematickými výrazmi, vytvárali sme funkcie, vedeli sme pomocou **for**-príkazu opakovať nejakú skupinu príkazov a ukázali sme si, ako vypísať niečo na obrazovku.

Aby sme však vedeli písať užitočné počítačové programy, potrebujeme taktiež mať možnost **meniť správanie programu podľa platnosti nejakých podmienok**, teda podľa toho, či je nejaká podmienka **pravdivá (True)** alebo **nepravdivá (False).**

Túto možnosť nám poskytujú tzv. **podmienené príkazy (conditional statements).**

Podmienené príkazy nám umožňujú meniť správanie programu podľa toho, či je splnená nejaká podmienka alebo nie. Často je podmienka spojená s dátami, ktoré program spracúva, teda pre rôzne vstupné dáta sa program môže správať rôznymi spôsobmi.

Podmienené príkazy sa niekedy nazývajú aj **vetvenie výpočtu**, pretože v závislosti na tom, či je splnená nejaká podmienka sa výpočet môže rozdeliť na disjunktné časti – ak podmienka splnená bola alebo nebola.

Neúplný podmienený príkaz (if)

Základným podmieneným príkazom je tzv. **neúplný podmienený príkaz**, známy ako **príkaz-if.**

Príkaz **if** je príkaz, ktorý skontroluje, či má nejaká podmienka hodnotu True alebo False – teda či **platí** alebo **neplatí**. Ak je podmienka pravdivá (má hodnotu *True*), tak sa vykoná nejaký príslušný blok príkazov. Ak podmienka pravdivá nie je, tak sa príslušný blok príkazov **nevykoná.**

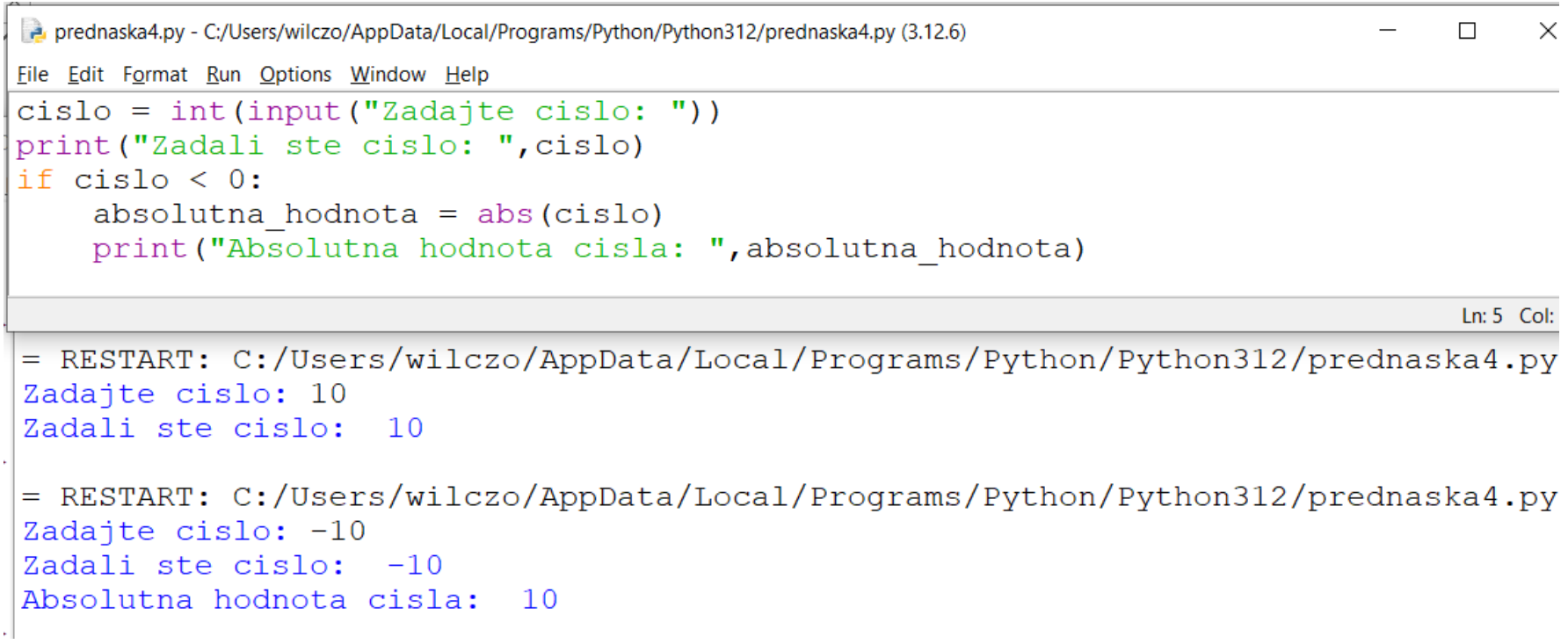
V jazyku Python sa neúplný podmienený príkaz implementuje nasledovným spôsobom

if *podmienka:*
    príkaz
    príkaz
    …
    príkaz

Tu je blok príkazov, ktoré sa vykonajú, ak je *podmienka* splnená teda *podmienka* je booleovský výraz s hodnotou *True*. Ak je *podmienka* nesplnená (má hodnotu *False)*, tak sa tento blok príkazov nevykoná.

Podmienené príkazy - if

Príklad programu s príkazom if. Program načíta číslo z klávesnice a ak je číslo záporné, program navyše vypíše jeho absolútnu hodnotu. Uvádzame 2 behy programu – raz sa zadalo číslo 10 a raz -10:



Úplný podmienený príkaz (if-else)

Úplným podmieneným príkazom je tzv. **príkaz-if-else.**

Príkaz **if-else** je príkaz, ktorý skontroluje, či má nejaká podmienka hodnotu True alebo False – teda či **platí** alebo **neplatí**. Ak je podmienka pravdivá (má hodnotu *True*), tak sa vykoná blok príkazov prislúchajúci **if-časti**. Ak podmienka pravdivá nie je, vykoná sa blok príkazov prislúchajúci **else-časti.**

if *podmienka:*
    príkaz
    príkaz
    …
    príkaz

Tu je **„if-časť“,** t.j. blok príkazov, ktoré sa vykonajú, ak je *podmienka* splnená teda *podmienka* je booleovský výraz s hodnotou *True*.

else:
    príkaz
    príkaz
    …
    príkaz

Tu je **„else-časť“,** t.j. blok príkazov, ktoré sa vykonajú, ak *podmienka* nie je splnená teda *podmienka* je booleovský výraz s hodnotou *False.*

Podmienené príkazy - if-else

Príklad programu s príkazom if-else. Program načíta číslo z klávesnice a **ak** je číslo **párne**, vypíše sa, že je číslo párne, **inak** sa vypíše, že číslo je nepárne. Uvádzame 2 behy programu – raz sa zadalo číslo 2 a raz 5:

prednaska4.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska4.py (3.12.6)

File Edit Format Run Options Window Help

```
cislo = int(input("Zadajte cislo: "))
if cislo %2 == 0:
    print("Zadali ste parne cislo!")
else:
    print("Zadali ste neparne cislo!")
```

```
= RESTART: C:/Users/wilczo/AppData/Local/Programs/Pyt
Zadajte cislo: 2
Zadali ste parne cislo!

= RESTART: C:/Users/wilczo/AppData/Local/Programs/Pyt
Zadajte cislo: 5
Zadali ste neparne cislo!
```

Podmienené príkazy - if-elif-else

Príkaz **if-else** teda umožňuje v programe vytvoriť 2 vetvy, ktorými sa môže výpočet uberať v závislosti na platnosti podmienky – preto sa podmienené príkazy nazývajú aj **vetvenie**. **Vždy** sa vykoná len 1 vetva! (lebo podmienka nemôže byť súčasne True aj False).

Avšak niekedy by sme chceli, aby sa program mohol rozdeliť na **väčší počet vetiev**, než dve. Napríklad uvažujme situáciu, že by sme chceli pre načítané číslo vypísať, či je:
- kladné
- záporné
- rovné nule (pretože nula nie je ani kladné ani záporné číslo)

Alebo by sme chceli porovnať 2 hodnoty *x* a *y* a vypísať, či je:
- *x* menšie ako *y*
- *x* väčšie ako *y*
- *x* rovné *y*

Teda v praxi môžu vzniknúť situácie, kde môže nastať viacero prípadov, než len 2. Preto potrebujeme aj v programe niekedy vetviť výpočet do viacerých, než len 2 vetiev.

Podmienené príkazy - if-elif-else

V prípade, že chceme vetviť výpočet na viacero vetiev, je to možné pomocou príkazu **if-elif-else**. Pomocou **if-elif-else** môžeme vo výpočte vytvoriť viacero výpočtových vetiev, podľa viacerých podmienok. Skratka **elif** znamená „**else if“.**

if *podmienka1:*
    blok príkazov
    ...
    #tu je blok príkazov, ktoré sa vykonajú ak je *podmienka1* True
elif *podmienka2:*
    #v prípade, že *podmienka1* je False, vyhodnotí sa *podmienka2*
    blok príkazov
    ...
    #tu je blok príkazov, ktoré sa vykonajú ak je *podmienka2* True
elif *podmienka3:*
    #v prípade, že podmienka2 je False, vyhodnotí sa *podmienka3*
    blok príkazov
    …
    #tu je blok príkazov, ktoré sa vykonajú ak je *podmienka3* True
…
    #príkazov **elif** môže byť ešte viac
else*:*
    blok príkazov
    ...
    #ak **žiadna** z podmienok nebola True, vykoná sa tento blok príkazov

Podmienené príkazy - if-elif-else

Program, ktorý načíta z klávesnice číslo a vypíše, či je 0, kladné alebo záporné. Ide teda o 3 rôzne prípady, teda potrebujeme použiť if-elif-else, aby sme vedeli pokryť všetky 3 prípady. Dole kód a 3 behy programu pre vstupy 5, 0 a -2.

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

File Edit Format Run Options Window Help

```
cislo = int(input("Zadajte cislo: "))
if cislo > 0:
    print("Cislo ", cislo, "je kladne")
elif cislo == 0:
    print("Cislo ", cislo, "je nula")
else:
    print("Cislo ", cislo, "je zaporne")
```

```
Zadajte cislo: 5
Cislo  5 je kladne

========== RESTART: Z:\Documents\p
Zadajte cislo: 0
Cislo  0 je nula

========== RESTART: Z:\Documents\p
Zadajte cislo: -2
Cislo  -2 je zaporne
```

Podmienené príkazy - if-elif-else

Ako to funguje?

```
cislo = int(input("Zadajte cislo: "))
if cislo > 0:
    print("Cislo ", cislo, "je kladne")
elif cislo == 0:
    print("Cislo ", cislo, "je nula")
else:
    print("Cislo ", cislo, "je zaporne")
```

1) Najprv sa vyhodnotí výraz cislo>0. Ak má hodnotu True, potom sa vykoná príslušný blok príkazov (výpis že je číslo kladné) a zvyšok if-elif-else sa nevykoná!

2) V prípade, že podmienka cislo>0 mala hodnotu False, pokračuje sa s testom podmienky cislo==0. Ak je táto podmienka True, tak sa vykoná príslušný blok príkazov (výpis, že číslo je nula) a zvyšok if-elif-else sa nevykoná.

3) V prípade, že podmienka cislo == 0 mala hodnotu False, pokračuje sa s else vetvou príkazu if-elif-else, teda sa vykonajú príkazy v else-časti. Vypíše sa teda, že číslo je záporné.

Príkaz if-elif-else sa teda vyhodnocuje od **prvej** podmienky pri **if.** Ak je táto podmienka splnená, vykoná sa príšlušný blok príkazov a program **ignoruje** zvyšok if-elif-else konštrukcie a pokračuje s ďalšími príkazmi, ktoré sú **za** celým if-elif-else blokom.

V prípade, že prvá podmienka nebola splnená, program pokračuje s najbližšou podmienkou – t.j. podmienka pri **elif** a postup sa zopakuje.

Vetva **else** sa vykoná, ak nebola True **žiadna** z podmienok.

Podmienené príkazy - if-elif-else

V príkazoch if, if-else, if-elif-else môžu byť ako podmienky, podľa ktorých sa vetví výpočet, použité aj zložené Booleovské výrazy, t.j. výrazy s logickými operátormi **and**, **or**, **not**.

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

File Edit Format Run Options Window Help

```
a = int(input("Zadajte velkost strany a: "))
b = int(input("Zadajte velkost strany b: "))
c = int(input("Zadajte velkost strany c: "))

if ((a+b) > c) and ((b+c) > a) and ((a+c) > b):
    print("Je mozne zostrojit truholnik so stranami dlzok: ",a,b,c)
else:
    print("Nie je mozne zostrojit truholnik so stranami dlzok: ",a,b,c)
```

```
Zadajte velkost strany a: 10
Zadajte velkost strany b: 5
Zadajte velkost strany c: 7
Je mozne zostrojit truholnik so stranami dlzok:  10 5 7
>>>
========== RESTART: Z:\Documents\prednaska\test.py ==========
Zadajte velkost strany a: 2
Zadajte velkost strany b: 3
Zadajte velkost strany c: 8
Nie je mozne zostrojit truholnik so stranami dlzok:  2 3 8
```

Vľavo príklad programu, ktorý načíta 3 čísla, ktoré by mali predstavovať strany trojuholníka a zistí, či spĺňajú trojuholníkovú nerovnosť.

Všimnite si zloženú podmienku – aby bola podmienka pravdivá, musia súčasne platiť všetky výrazy (a+b)>c, (b+c)>a, (a+c)>b.

Podmienené príkazy - if-elif-else

POZOR!!! V prípade if-elif-else platí, že ak nastane situácia, že by bolo splnených viacero podmienok, vykoná sa **vždy** len ten blok príkazov, ktorý prináleží **prvej splnenej podmienke!!!**

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

File Edit Format Run Options Window Help

```
cislo = int(input("Zadajte cislo: "))

if cislo % 2 == 0:
    print("Cislo ", cislo, "je delitelne dvojkou")
elif cislo % 3 == 0:
    print("Cislo ", cislo, "je delitelne trojkou")
else:
    print("Cislo nie je delitelne ani dvojkou, ani trojkou!")
```

```
Zadajte cislo: 10
Cislo  10 je delitelne dvojkou
>>>
========== RESTART: Z:\Documents\prednaska\test.py ==========
Zadajte cislo: 15
Cislo  15 je delitelne trojkou
>>>
========== RESTART: Z:\Documents\prednaska\test.py ==========
Zadajte cislo: 11
Cislo nie je delitelne ani dvojkou, ani trojkou!
>>>
========== RESTART: Z:\Documents\prednaska\test.py ==========
Zadajte cislo: 12
Cislo  12 je delitelne dvojkou
```

Vľavo je príklad programu, v ktorom ak je vstupom číslo deliteľné 2, tak sa vypíše, že je deliteľné 2 a zvyšok if-elif-else sa nevykoná!

Teda ak je na vstupe číslo, ktoré je deliteľné aj 2, aj 3, tak to, že je deliteľné 3 sa **nevypíše** (napr. pre číslo 12).

To, že je číslo deliteľné 3 sa vypíše **len pre také číslo**, ktoré je deliteľné 3, ale **nie je** deliteľné 2, teda prvá podmienka (cislo %2 == 0) bola preň **False**.

Podmienené príkazy - if-elif-else

POZOR!!! Niekedy sa dokonca môže stať, že podmienky pokryjú všetky možné prípady a vetva **else** sa nikdy nevykoná.

Nižšie je uvedený program, v ktorom sa vetva **else** nikdy nevykoná, pretože pre každé číslo je alebo splnená prvá podmienka (ak je kladné), alebo druhá podmienka (ak je 0 alebo záporné). Taktiež si všimnite, že existujú čísla, pre ktoré sú súčasne splnená aj podmienka (cislo>0), aj (cislo<10), avšak znovu platí, že ak platí prvá podmienka (cislo>0), tak sa vykoná príslušný príkaz (výpis, že je číslo kladné) a zvyšok if-elif-else sa nevykoná.

```
cislo = int(input())

if cislo > 0:
    print("Cislo ", cislo, "je kladne")
elif cislo < 10:
    print("Cislo ", cislo, "je mensie ako 10")
else:
    print("Tento vypis sa nikdy nevykona!")
```

Podmienené príkazy - if-elif-else

Počet **elif** vetiev v rámci príkazu if-elif-else je neobmedzený! Nižšie je príklad programu, ktorý pre počet bodov vypíše výslednú známku podľa známkovacej stupnice STU:

```
body = int(input("Zadajte pocet bodov: "))

if body >= 92:
    print("Znamka A")
elif body >= 83:
    print("Znamka B")
elif body >= 74:
    print("Znamka C")
elif body >= 65:
    print("Znamka D")
elif body >= 56:
    print("Znamka E")
else:
    print("Znamka FX")
```

Podmienené príkazy - if-elif-else

Nižšie je uvedená verzia predchádzajúceho programu, avšak s vymeneným poradím podmienok. Skúste si program spustiť pre 56 bodov – správne vypíše známku E. Skúste si program spustiť pre 55 bodov – správne vypíše FX. Ale čo vypíše pre napríklad 92 bodov?

```
body = int(input("Zadajte pocet bodov: "))

if body >= 56:
    print("Znamka E")
elif body >= 65:
    print("Znamka D")
elif body >= 74:
    print("Znamka C")
elif body >= 83:
    print("Znamka B")
elif body >= 92:
    print("Znamka A")
else:
    print("Znamka FX")
```

Keďže 92 bodov je viac ako 56, už **prvá podmienka** (body >= 56) je splnená, preto program vypíše *Znamka E* a zvyšné **elif** podmienky program nevyhodnotí.

Preto je **veľmi dôležité** pri použití príkazu if-elif-else dbať na štruktúru podmienok, teda v akom poradí sú zapísané pri **if** a **elif** príkazoch.

Podmienené príkazy - if-elif-else

Čo sa stane, ak v korektnom programe nahradíme **elif** príkazmi **if** ?

```
body = int(input("Zadajte pocet bodov: "))

if body >= 92:
    print("Znamka A")
if body >= 83:
    print("Znamka B")
if body >= 74:
    print("Znamka C")
if body >= 65:
    print("Znamka D")
if body >= 56:
    print("Znamka E")
else:
    print("Znamka FX")
```

```
========== RESTART: Z:\
Zadajte pocet bodov: 100
Znamka A
Znamka B
Znamka C
Znamka D
Znamka E

========== RESTART: Z:\
Zadajte pocet bodov: 70
Znamka D
Znamka E

========== RESTART: Z:\
Zadajte pocet bodov: 55
Znamka FX
```

Program teraz pozostáva z postupnosti **4** príkazov **if** (t.j. bez else časti) a jedného **if-else** príkazu!

Vnorené podmienené príkazy

Podmienené príkazy môžu byť taktiež súčasťou if/elif/else vetiev iných podmienených príkazov! Niekedy sa to nazýva aj **vnorené podmienené príkazy**.

```
cislo = int(input("Zadajte cislo x: "))

if cislo > 0:
    print ("Cislo ", cislo, "je kladne")
else:
    if cislo == 0:
        print ("Cislo ", cislo, "je nula")
    else:
        print ("Cislo ", cislo, "je zaporne")
```

1) ak je podmienka cislo>0 splnena, vykona sa if-cast, teda vypis, ze cislo je kladne.

2) ak podmienka cislo>0 nie je splnena, vykona sa else-cast, teda sa otestuje podmienka cislo==0

2.1) ak podmienka cislo==0 je splnena, vypise sa, ze cislo je nula

2.2) ak podmienka cislo==0 nie je splnena, vypise sa, ze cislo je zaporne

Správanie vyššie uvedeného programu je de facto identické, ako programu zo slajdov č. 36-37. Ak je splnená prvá podmiena, cislo>0, vykoná sa if-časť, teda vypís, že je číslo kladné.

Ak podmienka splnená nie je, vykoná sa else-časť, v rámci ktorej je znovu if-else príkaz, tentokrát s podmienkou cislo==0.

Vnorené podmienené príkazy

Niekedy sa dajú vnorené podmienené príkazy nahradiť pomocou logických operátorov. Napríklad program:

```
cislo = int(input("Zadajte cislo x: "))

if 0 < cislo:
    if cislo < 10:
        print("Cislo je kladne a jednociferne")
```

kde k výpisu správy „*Cislo je kladne a jednociferne*“ dôjde len vtedy, keď je splnená aj podmienka *0<cislo*, aj podmienka *cislo<10*. Teda musia byť splnené obe podmienky, čo vieme jednoduchšie zapísať pomocou logického operátora **and**:

```
cislo = int(input("Zadajte cislo x: "))

if 0 < cislo and cislo < 10:
    print("Cislo je kladne a jednociferne")
```

Dôležité príklady

Pre zopakovanie, dôležitý príklad z 2. prednášky

Funkcia, ktorá vráti súčet čísiel od 1 po *n*, kde *n* je parameter funkcie.

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

Tento príklad je dôležitý, aby ste si všimli, ako je naprogramované to, že potrebujeme postupne sčítať nejaké čísla dokopy. Tu je to riešené pomocou premennej *sucet*, do ktorej postupne pripočítame jednotlivé hodnoty, ktoré chceme sčítať.

Navyše je nutné premennú pred jej prvým použitím v cykle inicializovať na 0, aby v momente, keď vykonáme príkaz
sucet = sucet + (i+1)
premenná *sucet* už obsahovala nejakú hodnotu a výraz na pravej strane priradenia sa dal vyhodnotiť a vložiť do premennej *sucet.*

Pre zopakovanie, dôležitý príklad z 2. prednášky

To znamená, ak chceme v programe zistiť súčet čísiel, ktoré majú nejakú vlastnosť, môžeme postupovať nasledovne:

1) Vytvoríme si novú premennú (napr. s názvom *sucet)* , ktorú inicializujeme na 0

2) Postupne generujeme hodnoty, ktoré chceme sčítať a každú takúto hodnotu pripočítame k premennej *sucet* štýlom: *sucet = sucet + hodnota*

3) Po skončení generovania hodnôt sa v premennej *sucet* bude nachádzať hľadaný výsledný súčet.

4) To, čo následne s premennou *sucet* urobíme už závisí na type úlohy (výpis, vrátenie z funkcie, atď.)

Druhý dôležitý príklad

Funkcia s parametrom celé číslo *n*, ktorá načíta *n* čísiel z klávesnice a vráti **počet** načítaných čísiel, ktoré boli párne.

```
def pocet_parnych(n):
    pocitadlo = 0
    for i in range(n):
        nacitane_cislo = int(input())
        if nacitane_cislo % 2 == 0:
            pocitadlo = pocitadlo + 1
    return pocitadlo
```

Tento príklad je dôležitý, aby ste si všimli, ako spočítame **koľko** hodnôt malo nejakú hľadanú vlastnosť. Vytvoríme si pomocnú premennú *pocitadlo*, ktorú nastavíme na nula a vždy, keď nájdeme hodnotu s požadovanou vlastnosťou, tak premennú *pocitadlo* zvýšime o 1.

Na záver sa v nej nachádza počet, koľkokrát sme videli hodnoty s požadovanou vlastnosťou (v tomto prípade či bolo načítané číslo párne, čo zistíme tak, že párne číslo má po delení 2 zvyšok 0, teda ak je načítané číslo párne, tak (*nacitale_cislo % 2 == 0)* má hodnotu True.

Druhý dôležitý príklad

To znamená, ak chceme v programe zistiť počet čísiel, ktoré majú nejakú vlastnosť, môžeme postupovať nasledovne:

1) Vytvoríme si novú premennú (napr. s názvom *pocitadlo)* , ktorú inicializujeme na 0, pretože **doterajší počet hodnôt** s požadovanou vlastnosťou bol nula.

2) Postupne sledujeme požadovanú vlastnosť pre rôzne hodnoty a ak zistíme, že hodnota má požadovanú vlastnosť, tak zvýšime *pocitadlo* o 1 spôsobom:
pocitadlo = pocitadlo + 1

3) To, či má hodnota požadovanú vlastnosť testujeme pomocou vhodne zvoleného **if**

4) Po skončení hodnôt sa v premennej *pocitadlo* bude nachádzať hľadaný výsledný počet hodnôt, ktoré mali sledovanú vlastnosť.

Tretí dôležitý príklad

Funkcia s parametrom celé číslo *n*, ktorá načíta *n* čísiel z klávesnice a vráti **najväčšie** načítané číslo

```
def max_cislo(n):
    maximum = int(input())
    for i in range(n-1):
        nacitane_cislo = int(input())
        if nacitane_cislo > maximum:
            maximum = nacitane_cislo
    return maximum
```

Program funguje tak, že v premennej *maximum* máme uloženú tú hodnotu, ktorá bola najväčšia z **doteraz načítaných čísiel**. Preto do nej uložíme **prvú načítanú hodnotu** a zvyšných n-1 hodnôt načítame v cykle. Ak zistíme, že sa načítala z klávesnice taká hodnota, že je ešte väčšia než *maximum*, znamená to, že sme práve načítali ešte väčšie číslo, než predtým a teda sme objavili nové maximum.

POZOR!!! Inicializácia premennej *maximum* na hodnotu nula je **zlé riešenie** pretože čo ak by všetky načítané hodnoty boli záporné???

Tretí dôležitý príklad

Tretí príklad je dôležitý, aby ste si všimli, ako algoritmicky hľadáme hodnotu, ktorá má nejakú špeciálnu vlastnosť **v porovnaní s ostatnými hodnotami.**

Preto postupujeme tak, že si pamätáme v pomocnej premennej tú hodnotu, ktorá **doteraz** mala túto vlastnosť najlepšiu – napríklad v prípade hľadania maxima je tam uložená hodnota, ktorá bola **doteraz** najväčšia.

Ak by sme potrebovali pri sledovaní nejakej vlastnosti pracovať s viacerými hodnotami, je vhodné si pre každú takúto hodnotu vytvoriť samostatnú premennú (na cvičení bude príklad s hľadaním **druhého najväčšieho čísla**, tam si musím pamätať 2 čísla!)

Zároveň si je nutné uvedomiť, ako túto špeciálnu hodnotu inicializovať! Ak hľadáme maximum spomedzi načítaných čísiel, tak na začiatku je to určite **prvé načítané číslo**.