# Oznam 1

Tento štvrtok je študijné voľno.

Študenti, ktorí majú cvičenia v štvrtok, sa tento týždeň môžu zúčastniť cvičenia v stredu v čase o 15:00 alebo 17:00.

Pre študentov, ktorí majú cvičenie vo štvrtok, ale nie je účasť na cvičení tento týždeň povinná.

Výsledky z prvého testu budú zapísané do AISu budúci týždeň, taktiež písomky k nahliadnutiu budú k dispozícii na cvičeniach budúci týždeň.

# PROG1: Prednáška 8
## Reťazce (strings)

*Why don’t strings make good secret agents?*

*Because they always get caught in quotes!*

*© ChatGPT*

# Reťazce

Reťazec (string) je *dátový typ* – teda typ hodnoty, ktorú môžu nadobúdať dáta.

Už na prvej prednáške sme si reťazce spomínali – reťazec je **usporiadaná postupnosť znakov.**

Doteraz sme pri programovaní využívali premenné, ktoré mohli obsahovať len atomické hodnoty – celé čísla (int) alebo desatinné čísla (float).

Reťazce sú príkladom tzv. štruktúrovaného dátového typu, t.j. 1 hodnota (1 reťazec) je **zoskupením** jednoduchších dátových typov (v tomto prípade jednotlivých znakov).

Reťazce sú príkladom **homogénneho** štruktúrovaného dátového typu, keďže všetky jednoduché hodnoty, z ktorých pozostáva, sú rovnakého typu (v tomto prípade textové znaky).

# Reťazce

S reťazcami sme sa už doteraz stretli:

1) reťazce sú tvorené postupnosťou textových znakov, ktoré sú uzatvorené do:
   - apostrofov ‘ ‘
   - úvodzoviek “ “

2) keďže reťazce sú hodnoty, môžeme ich vkladať do premenných a vykonávať s nimi rôzne operácie – na prvej prednáške sme si spomínali napríklad zreťazenie (+) reťazcov, alebo násobenie reťazca číslom

# Reťazce

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

File Edit Format Run Options Window Help

```
a = "ahoj"        #retazec "ahoj" uzatvoreny do uvodzoviek ""
b = 'studenti'    #retazec 'studenti' uzatvoreny do apostrofov ''
print(a)          #vypis retazca v premennej a na obrazovku
print(b)          #vypis retazca v premennej b na obrazovku
print(a+b)        #vypis vysledku operacie zretazenia (+) na obrazovku
print(a*3)        #vypis vysledoku operacie viacnasobneho zretazenia na obrazovku
```

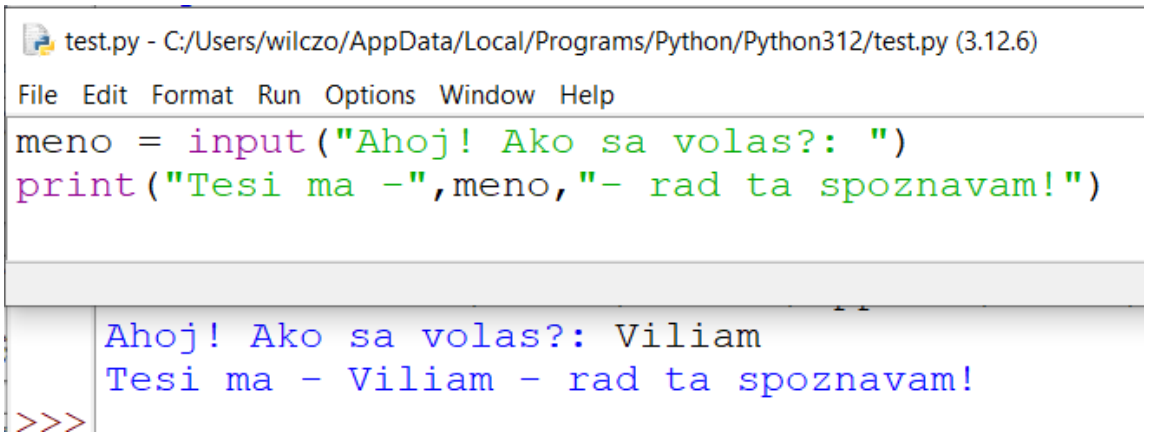
Ln: 7 Col

```
ahoj
studenti
ahojstudenti
ahojahojahoj
```

# Reťazce

Ak chceme načítať reťazec z klávesnice od používateľa, slúži na to priamo funkcia **input().**

Funkcia **input()** priamo **vráti** načítaný vstup od používateľa ako **reťazec**.




Len pripomínam, že keď sme chceli načítať z klávesnice **celé číslo**, tak sme museli návratovú hodnotu **input()** ešte vložiť do funkcie **int(), t.j. int(input())**, aby sa načítaná hodnota interpretovala ako celé číslo. V prípade reťazcov to **nerobíme**, ale stačí len volanie **input().**

# Reťazce

Reťazec je štruktúrovaný dátový typ pozostávajúci z textových znakov.

Okrem toho, že vieme pracovať s celým reťazcom, vieme pracovať aj s jednotlivými znakmi, ktoré ho tvoria. Konkrétne vieme získať ich hodnotu pomocou operátora *hranaté zátvorky (angl. brackets)*, do ktorých vložíme **index** znaku v reťazci, ktorého hodnotu chceme získať.

**Index** je poradové číslo znaku v reťazci – poradové čísla hodnôt v štruktúrovaných dátových typoch sa v informatike spravidla **indexujú (číslujú) od NULY!**

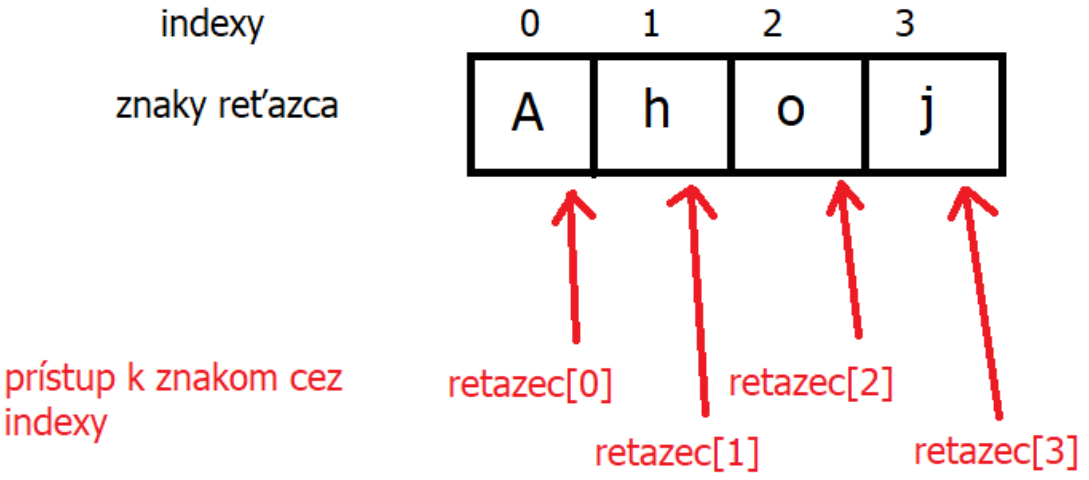
# Reťazce

test.py - C:/Users/wilczo/AppData/Local/Programs/Pytho

File Edit Format Run Options Window Help

```
retazec = "Ahoj"
print(retazec[0])
print(retazec[1])
print(retazec[2])
print(retazec[3])
```

```
A
h
o
j
```



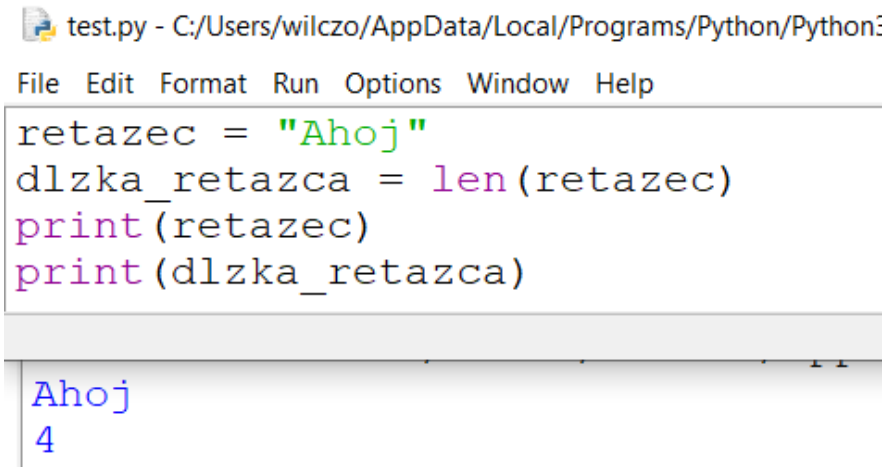
# Dĺžka reťazca

Veľa programovacích jazykov ponúka funkcie, ako zistiť, akú má reťazec dĺžku.

Teda, **koľko znakov obsahuje**.

V jazyku Python na to slúži vstavaná funkcia **len()**.

V príklade vytvoríme reťazec „Ahoj“, ktorý obsahuje 4 znaky: ‘A‘, ‘h‘, ‘o‘, ‘j‘



# Dĺžka reťazca

Keďže znaky v reťazci sú indexované od 0, to znamená, že **posledný znak** v reťazci uloženom v premennej *retazec* má index **len(***retazec***)**-1

Napríklad ak je v premennej *retazec* = *“Ahoj“*, teda **len(***retazec***)** vráti **4**, potom posledný znak “ *j* “ má v reťazci index 3.

Čo sa stane v Pythone, ak by sme sa pokúsili o prístup k znaku na indexe **väčšom** ako je index posledného znaku???

# Dĺžka reťazca

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

File Edit Format Run Options Window Help

```
retazec = "Ahoj"
print(retazec[0])
print(retazec[1])
print(retazec[2])
print(retazec[3])
print(retazec[4])
```

Tieto 4 znaky sa vypíšu na obrazovku

Pri pokuse o výpis znaku na indexe MIMO rozsah reťazca vypíše Python IndexError!

Ln: 6 Col: 15

```
A
h
o
j
Traceback (most recent call last):
  File "C:/Users/wilczo/AppData/Local/Programs/Python/Python312/test.py", line 6, in <module>
    print(retazec[4])
IndexError: string index out of range
```

# Indexácia „odzadu“

V jazyku Python je však možné pri indexácii znakov použiť aj **záporné hodnoty indexov!**
V takom prípade sú pristupované prvky **odzadu**, t.j. od posledného znaku k prvému.
Posledný znak má index -1, predposledný znak -2, atď.:

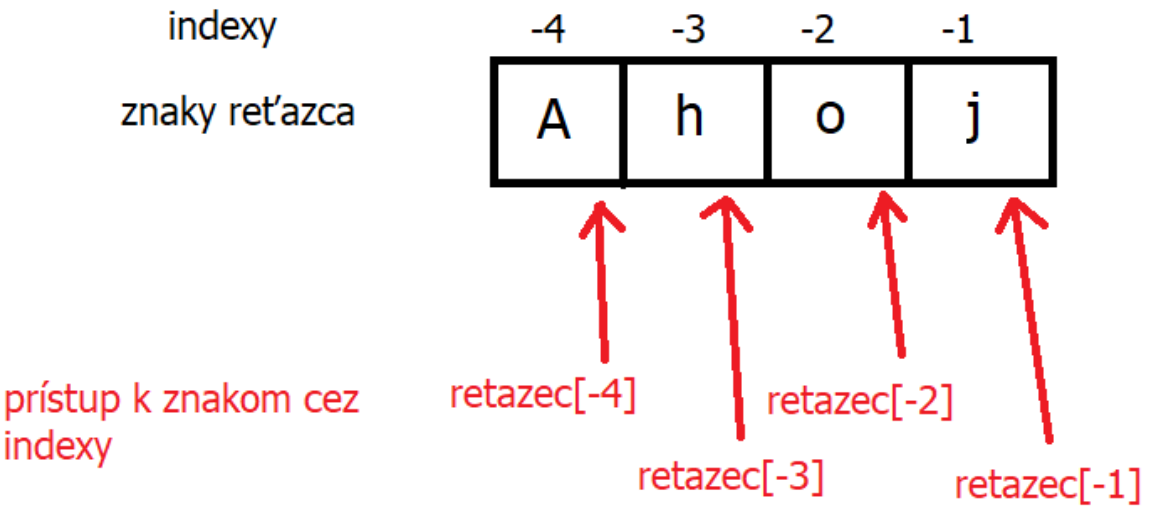
test.py - C:/Users/wilczo/AppDat...

File Edit Format Run Options Window Help

```
retazec = "Ahoj"
print(retazec[-1])
print(retazec[-2])
print(retazec[-3])
print(retazec[-4])
```

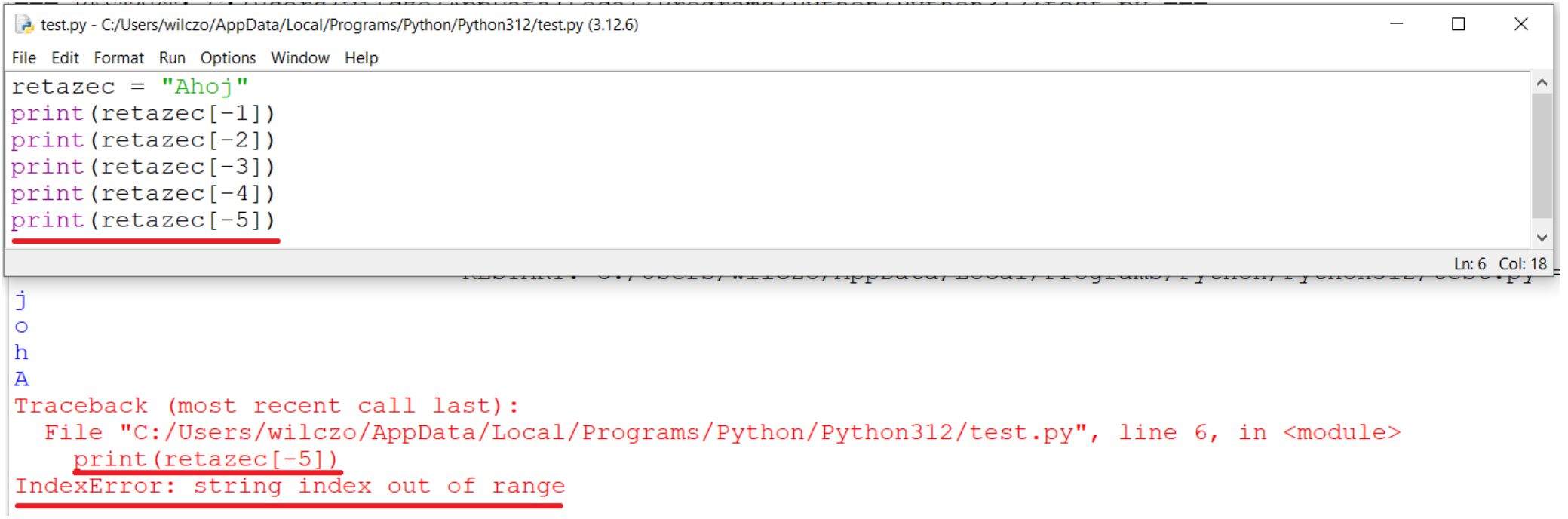
Ln: 6 Col: 0

```
j
o
h
A
```



# Indexácia „odzadu“

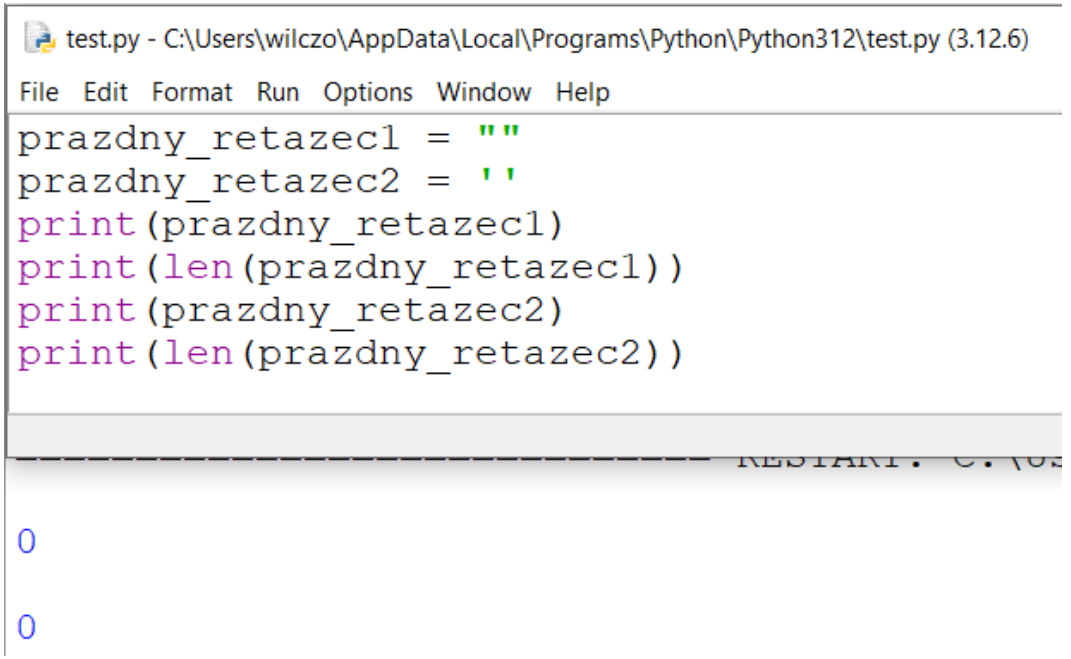
Aj pri indexácii odzadu nás Python upozorní, ak použijeme index mimo rozsah:



# Prázdny reťazec

Existuje aj **prázdny reťazec**, t.j. reťazec, ktorý neobsahuje žiadne znaky a jeho dĺžka je teda **nula**.

Vytvoríme ho ako “” alebo ‘’.



# Prechádzanie znakov reťazca (traversing)

V programovaní často potrebujeme prechádzať (prehľadávať) jednotlivé elementy štruktúrovaných dátových typov – tento proces sa angl. nazýva **traversing.** Štandardne sa začne s prvým elementom a postupne sa prechádza cez jednotlivé elementy, až po posledný.

V prípade reťazcov teda budeme takto postupne prechádzať **znaky reťazca.**

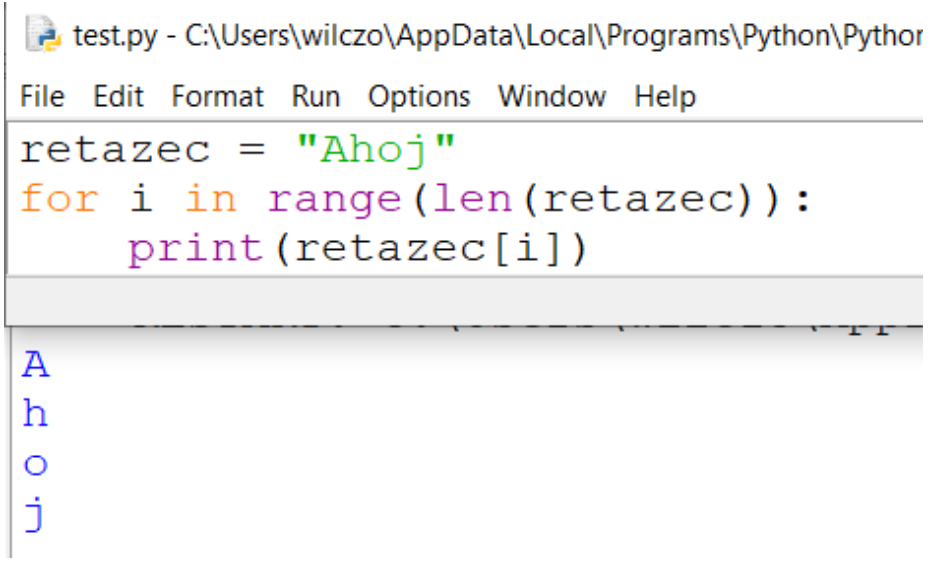
V jazyku Python je možné prechádzať znaky reťazca 2 spôsobmi:

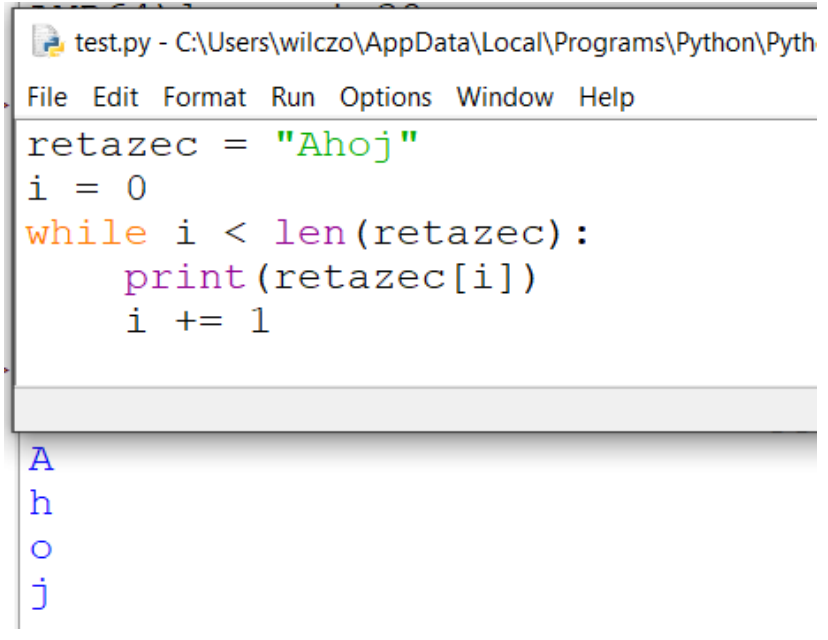
a) cez **indexy** prvkov reťazca

b) **priamo cez prvky** reťazca

# Prechádzanie znakov reťazca pomocou indexov

Pomocou cyklu (**while** alebo **for**) si môžeme elegantne postupne generovať indexy pre prístup k jednotlivým znakom. V oboch uvedených prípadoch vytvoríme premennú *i* , ktorú budeme používať ako index pri prístupe k znakom, všimnite si, že bude postupne nadobúdať hodnoty od 0 po **len**(retazec)-1, vrátane. Vľavo príklad pomocou **for**-cyklu, vpravo **while-**cyklus.

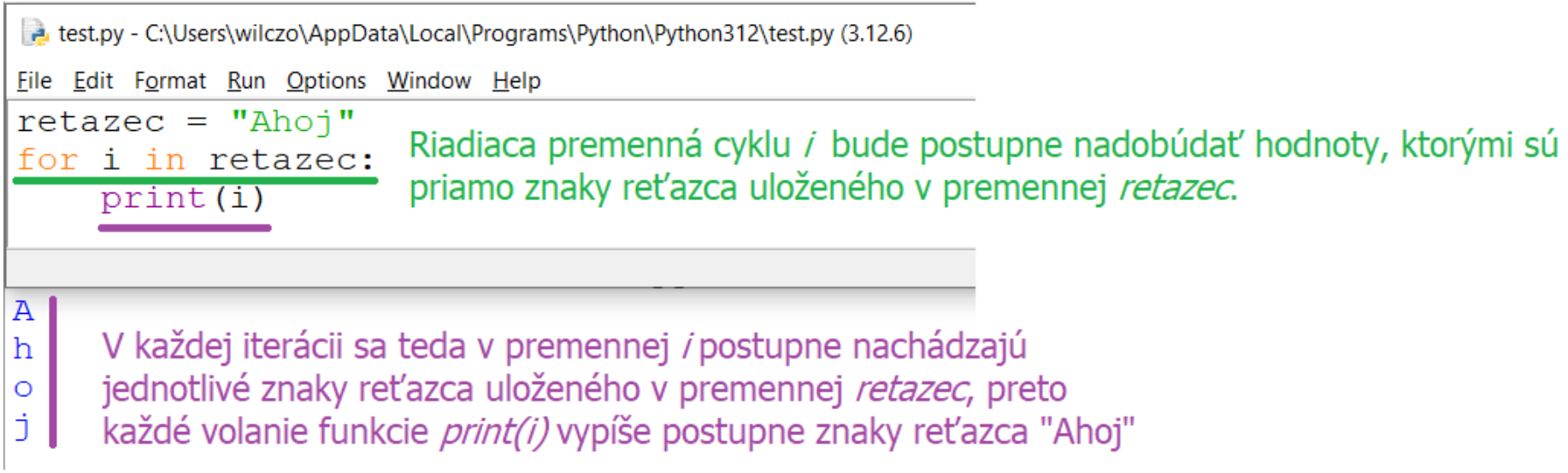


# Prechádzanie znakov reťazca priamo

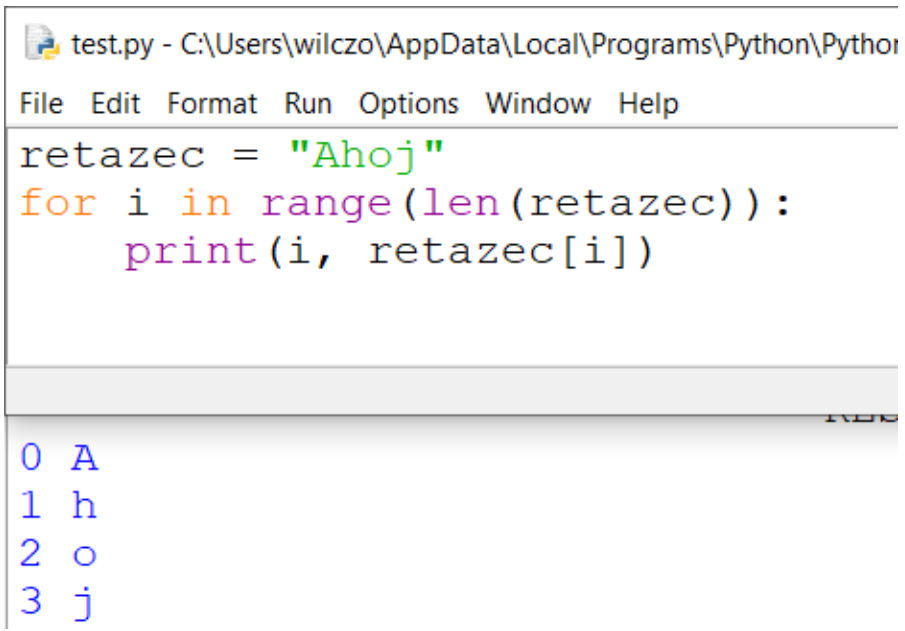
Niektoré programovacie jazyky, vrátane Pythonu, poskytujú možnosť prechádzať cez elementy štruktúrovaného dátového typu **priamo**. V tomto prípade to znamená, že sa vytvorí premenná, ktorá bude **priamo** nadobúdať hodnoty jednotlivých **znakov reťazca**.

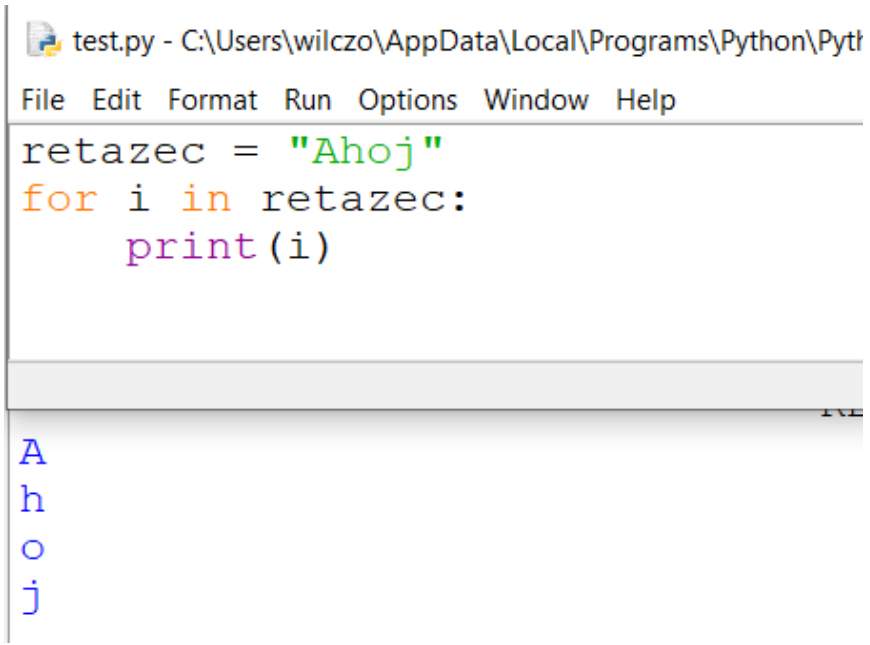
V jazyku Python má uvedený prístup nasledovnú syntax pomocou príkazu **for**:



# Porovnajte oba spôsoby!

Dobre si všimnite rozdiely medzi prechodom cez znaky reťazca:
a) vľavo pomocou indexov znakov v reťazci
b) vpravo priamo cez znaky reťazca



# Ktorý spôsob použiť?

Oba spôsoby sú v princípe zameniteľné, konkrétne použitie závisí na type úlohy, ktorú chceme riešiť. Napríklad predpokladajme, že by sme chceli definovať funkciu *najdi(retazec,znak),* ktorá vráti index prvého výskytu znaku *znak* v reťazci *retazec*. Ak sa *znak* v *retazec* nenachádza, funkcia vráti -1.

test.py - C:\Users\wilczo\AppData\Local\Programs\Python\Python312\test.py (3.12.6)

File Edit Format Run Options Window Help

```
def najdi(retazec, znak):
    for i in range(len(retazec)):
        if znak == retazec[i]:
            return i
    return -1
```

V riešení uvedenom vľavo využijeme prechádzanie reťazcom pomocou **indexov**, pretože aj úloha samotná vyžaduje, aby sme našli index hľadaného znaku.

Zároveň si všimnite, že pri prvom výskyte *znak* v *retazec* je splnená podmienka v **if** príkaze a teda sa vykoná **return i**, čím je zabezpečené, že sa vráti index prvého výskytu. V prípade, že sa *znak* nenachádza v reťazci, **nikdy** sa v cykle **return i** nevykoná, cyklus skončí a vykoná sa **return -1.**

# Výseky z reťazca

Jazyk Python okrem výberu konkrétneho znaku z reťazca pomocou indexácie poskytuje možnosť aj výberu **podreťazca**, t.j. postupnosti viacerých znakov, pomocou tzv. *výsekov* (angl. *slice)*.

Operátor výseku (slice) je tiež tvorený hranatými zátvorkami [ ], avšak v tomto prípade si môže používateľ vybrať:
a) index **začiatku** výberu *n*
b) index **konca** výberu *m*
c) **krok** výberu *k*

Ak by bol reťazec uložený v premennej *retazec*, potom postupnosť znakov **začínajúcu na** indexe *n*, **končiacu pred** indexom *m* (t.j. nie vrátane *m*) s **krokom** *k* vyberieme pomocou operátora výseku ako:

*retazec*[*n*:*m*:*k*]

# Výseky z reťazca

Parametre operátora výseku **nie sú povinné**. V prípade, že:

a) nie je uvedený parameter začiatku výseku, berie sa výsek **od začiatku reťazca**
*retazec[:m:k]* - výsek od začiatku po index *m* (nie vrátane) s krokom *k*

b) nie je uvedený parameter konca výseku, berie sa výsek **po koniec reťazca**
*retazec[n::k]* - výsek od indexu *n* po koniec reťazca s krokom *k*

c) nie je uvedený parameter kroku, berie sa výsek s **krokom 1**
*retazec[n:m]* - výsek od indexu *n* po index *m* (nie vrátane) s krokom 1

d) sú možné aj rôzne kombinácie:
*retazec[n:]* - výsek od indexu *n* po koniec reťazca s krokom 1
*retazec[:m]* - výsek od začiatku po index *m* (nie vrátane) s krokom 1
*retazec[:]* - výsek od začiatku po koniec s **krokom 1** (t.j. vráti ten istý reťazec)
*retazec[::-1]* - výsek od začiatku po koniec s **krokom -1** (t.j. vráti **obrátený reťazec**).

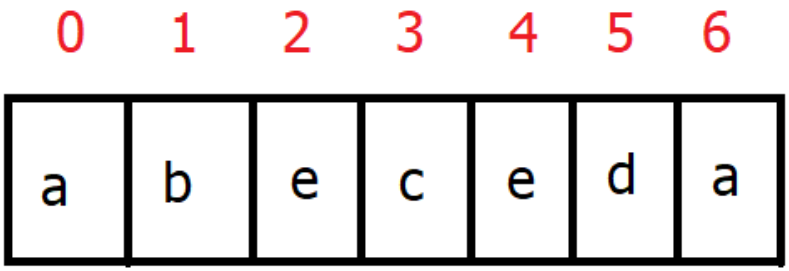
# Výseky z reťazca - príklady

test.py - C:\Users\wilczo\AppData\Local\Programs\Python\Python312\test.py (3.12.6)

File Edit Format Run Options Window Help

```
retazec = "abeceda"
print(retazec[1:5:1])
print(retazec[1:5])
print(retazec[1:5:2])
print(retazec[5:1:-1])
print(retazec[1:])
print(retazec[:4])
print(retazec[1::2])
print(retazec[5::-2])
```

```
bece
bece
bc
dece
beceda
abec
bcd
dcb
```

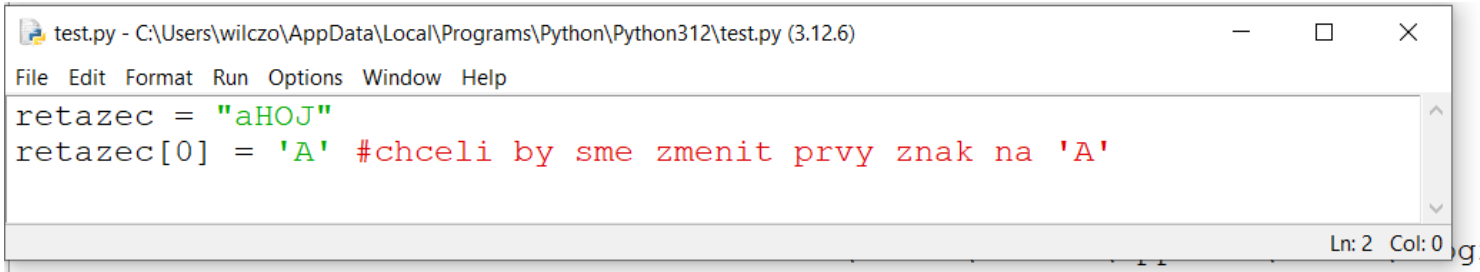


# Zmena znakov v reťazci

Keďže reťazec je usporiadaná postupnosť znakov, je prirodzená otázka, či vieme reťazce meniť.

V mnohých programovacích jazykoch je možné pomocou prístupu cez indexy meniť znaky reťazca. V jazyku Python to však **nie je možné**, pretože v jazyku Python je každý reťazec po jeho vytvorení **nemenný (immutable).**

V príklade vidíme, že sa nám nepodarí priamo zmeniť prvý znak reťazca v premennej *retazec* na veľké „A“:

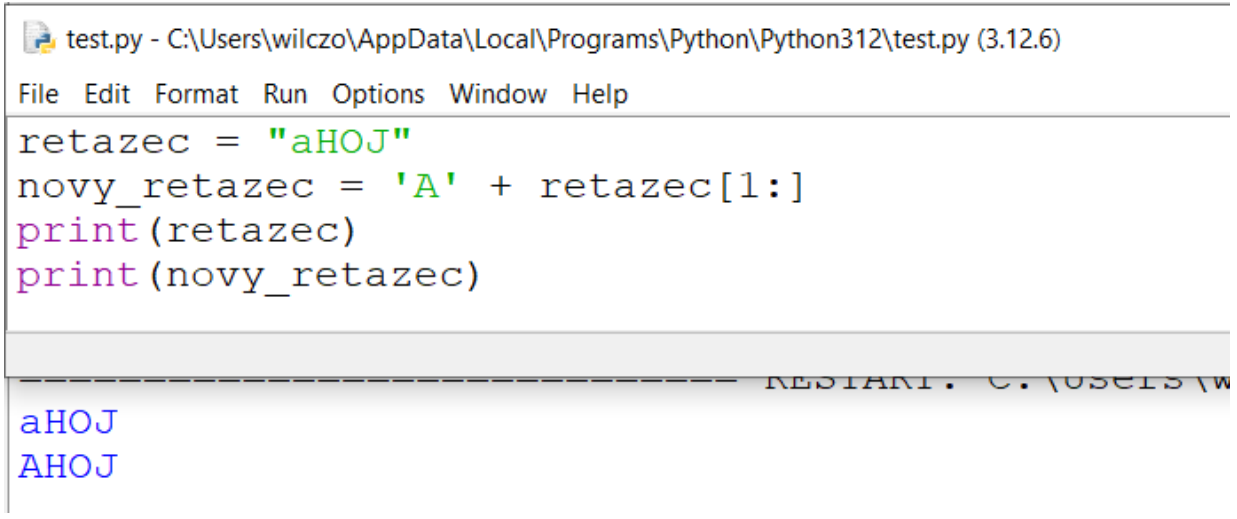



```
grams\Python\Python312
Traceback (most recent call last):
  File "C:\Users\wilczo\AppData\Local\Programs\Python\Python312\test.py", line 2, in <module>
    retazec[0] = 'A' #chceli by sme zmenit prvy znak na 'A'
TypeError: 'str' object does not support item assignment
```

# Zmena znakov v reťazci

Ak chceme zmeniť znak v reťazci, musíme **vytvoriť nový reťazec**, ktorý bude túto zmenu obsahovať.

V príklade vytvoríme nový reťazec, ktorý vznikne tak, že zreťazíme znak, ktorým sme chceli nahradiť prvý znak v premennej *retazec* so zvyškom reťazca *retazec.* Reťazec v premennej *retazec* teda zostal „aHOJ“ a v premenej *novy_retazec* sme vytvorili „AHOJ“:



# Metódy pre reťazce

Jazyk Python poskytuje viacero vstavaných funkcií, ktoré umožňujú vykonávanie rôznych operácií s reťazcami (hľadanie znakov/podreťazcov v reťazci, počet výskytov znakov/podreťazca, rôzne testy či sú všetky znaky v reťazci veľké písmená/malé písmená/číslice/textové znaky, atď.)

Tieto funkcie, ktoré sú relevantné len pre nejaký dátový typ – v tomto prípade reťazce – nazývame **metódy.**

Samotný pojem metóda súvisí s tzv. objektovo-orientovaným programovaním (reťazce v jazyku Python predstavujú inštancie triedy „str“ (string)) – o tom sa viac dozviete na predmete B-OOP (objektovo-orientované programovanie).

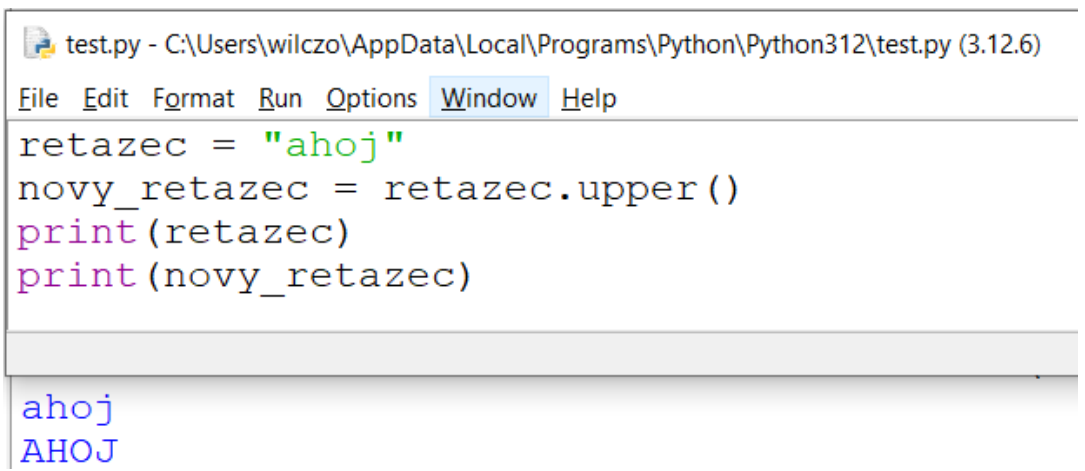
Nám na predmete B-PROG1 bude zatiaľ stačiť vedieť iba to, že v Pythone existujú takéto funkcie (metódy), ktoré vieme používať na reťazce, pričom **syntax** ich použitia je trochu iná, než syntax používania klasických funkcií.

# Metódy pre reťazce

Napríklad v Pythone existuje metóda **upper()**, ktorá **vráti** verziu reťazca, ktorá má všetky písmená **veľké**.

Aby sme túto metódu použili, musíme použiť tzv. **bodka-notáciu (dot-notation)**, ktorou sa používajú metódy. Ak je reťazec, na ktorý chceme metódu použiť, uložený v premennej *retazec*, potom metódu **upper()** pomocou bodka-notácie voláme ako:

*retazec*.**upper()**




Všimnite si, že **nedošlo k zmene** pôvodného reťazca, t.j. v premennej *retazec* je stále reťazec “ahoj“ s malými písmenami.

Ale v novej premennej je už uložený kapitalizovaný reťazec “AHOJ“

# Metódy pre reťazce

Ďalšou zaujímavou metódou je metóda **find()**. Metóda **find()** má jeden povinný parameter, ktorým je reťazec *sub.* Ak je metóda volaná pre reťazec *retazec* ako *retazec.find(sub)*, tak vráti pozíciu (index) v reťazci *retazec*, od ktorej sa v ňom nachádza ako podreťazec reťazec *sub*. Ak sa *sub* nenachádza ako podreťazec v reťazci *retazec*, tak vráti -1.

Metóda **find()** má navyše 2 nepovinné parametre *start* a *end*, ktoré umožnia špecifikovať, odkiaľ-pokiaľ sa má hľadať výskyt podreťazca *sub* v reťazci *retazec*, pre ktorý metódu voláme. V podstate

*retazec.find(sub, start, end)*

funguje rovnako ako

*retazec[start:end].find(sub)*

# Metódy pre reťazce - find

test.py - C:\Users\wilczo\AppData\Local\Programs\Python\Python312\test.py (3.12.6)

File Edit Format Run Options Window Help

```
retazec = "abeceda"
print(retazec.find("bec")) #vrati index, kde sa v retazec nachadza "bec" ako podretazec
print(retazec.find("c"))   #vrati index, kde sa v retazec nachadza "c" ako podretazec
print(retazec.find("c",0,2)) #vrati index, kde sa v retazec[0:2] nachadza "c"
print(retazec.find("f"))   #vrati index, kde sa v retazec nachadza "f"
```

```
1
3
-1
-1
```

# Metódy pre reťazce

- Na webstránke

  https://docs.python.org/3/library/stdtypes.html#string-methods

  je možné dočítať sa viac o rôznych metódach, ktoré je možné používať pri práci s reťazcami.

# Operátor **in**

V programovaní je často úlohou zistiť, či sa v nejakom štruktúrovanom dátovom type vyskytuje nejaká hodnota. Napríklad v prípade reťazcov zistiť, či obsahuje nejaký konkrétny znak alebo podreťazec.

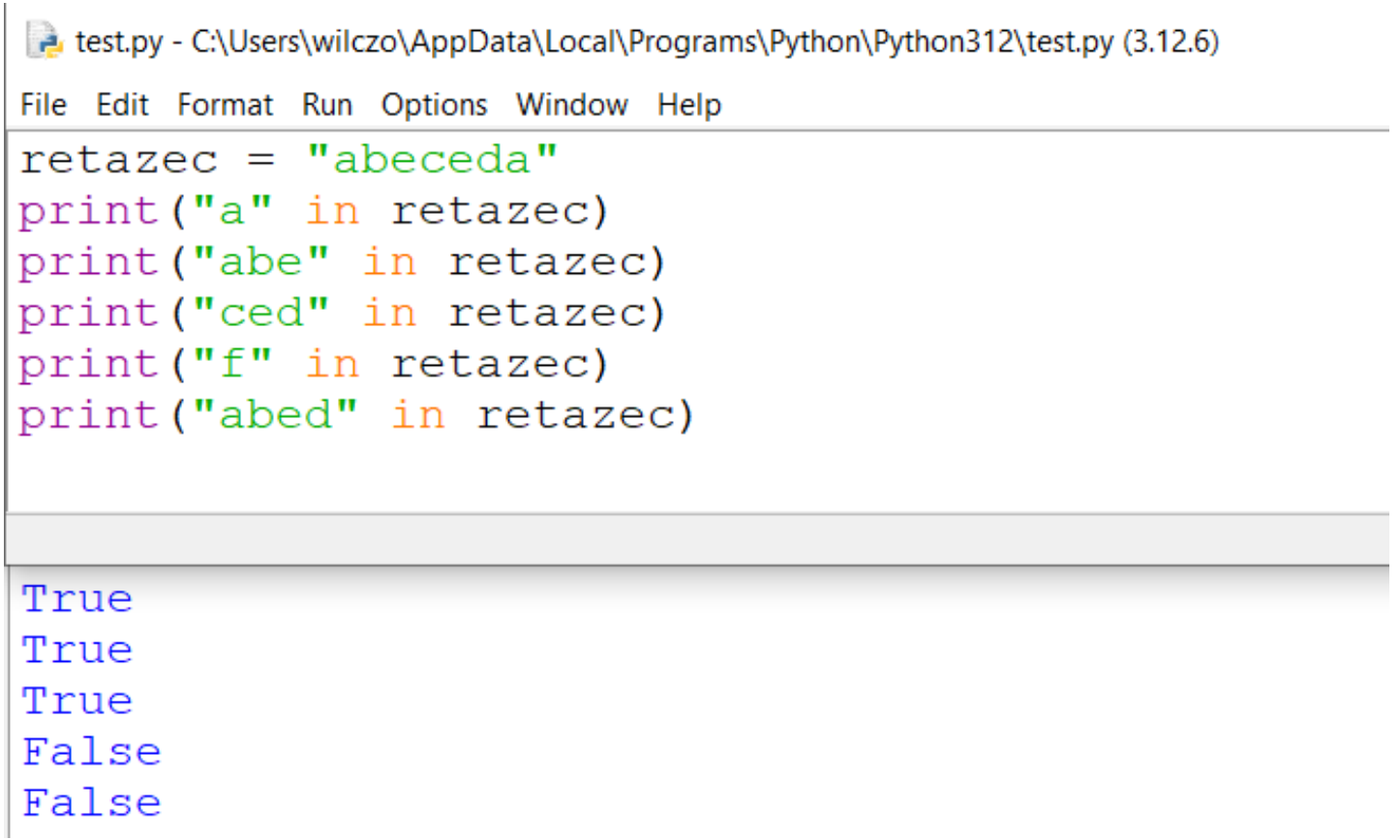
V predošlých slajdoch sme si ukázali, že Python obsahuje metódu **find**() ktorá vracia index, kde sa hľadaný podreťazec nachádza.

V prípade, že nás zaujíma len to, či sa v danom reťazci nejaký podreťazec nachádza alebo nie, tak v jazyku Python môžeme použiť operátor **in**. Operátor **in** je operátor v jazyku Python, ktorý vráti *True / False* podľa toho, či sa nejaký hľadaný *podretazec* nachádza v reťazci *retazec*, pričom syntax použitia je:

*podretazec* **in** *retazec*

a výsledok operátora je hodnota *True* / *False*.

# Operátor **in**


test.py - C:\Users\wilczo\AppData\Local\Programs\Python\Python312\test.py (3.12.6)
File Edit Format Run Options Window Help
retazec = "abeceda"
print("a" in retazec)
print("abe" in retazec)
print("ced" in retazec)
print("f" in retazec)
print("abed" in retazec)
True
True
True
False
False

# Operátor **in**

V Pythone teda existuje operátor **in**, ktorý má dvojaké využitie:

a) používa sa v cykle **for** na iterovaný prechod cez štruktúrovaný dátový typ:
- prechod cez symboly reťazca:

```
for znak in retazec:
    #premenná znak nadobúda hodnoty priamo znakov z reťazca
```

- prechod cez indexy symbolov reťazca:

```
for i in range(len(retazec)):
    #premenná i nadobúda hodnoty indexov z množiny {0, …, len(retazec)-1}.
```

b) používa sa ako booleovský operátor na zistenie, či sa nejaký podreťazec (prípadne znak) nachádza v nejakom reťazci

```
podretazec in retazec   #nadobúda hodnoty True / False podľa situácie
```

# Operátor **in - príklad**

Príklad: Definujte funkciu *v_oboch(retazec1, retazec2)* s 2 parametrami, ktorými sú reťazce *retazec1* a *retazec2*, ktorá vypíše **všetky** znaky z reťazca *retazec1*, ktoré sa nachádzajú aj v reťazci *retazec2.*

Analýza:
1) Keďže chceme vypísať **každý** znak z reťazca *retazec1*, pre ktorý má niečo platiť – v tomto prípade to, či sa nachádza zároveň aj v reťazci *retazec2*, tak zostrojíme cyklus, v ktorom budeme postupne prechádzať **všetky** znaky *retazec1* a pre každý znak otestujeme to, či sa nachádza v reťazci *retazec2*

2) Na prechod cez **všetky** znaky *retazec1* môžeme použiť napríklad priamy prechod cez znaky reťazca *retazec1*, t.j.

    **for** *znak* **in** *retazec1*:

(pokračovanie na ďalšom slajde)

# Operátor **in - príklad**

Analýza (pokračovanie)

3) Následne premenná *znak* bude postupne nadobúdať hodnoty znakov z reťazca *retazec1*. Keďže potrebujeme každý takýto znak otestovať na prítomnosť v reťazci *retazec2*, môžeme to jednoducho urobiť pomocou operátora **in** ako:

*znak* **in** *retazec2*

4) V prípade, že (*znak* **in** *retazec2)* nadobudne hodnotu *True* vieme, že sa aktuálna hodnota v premennej *znak*, ktorou je nejaký znak z reťazca *retazec1*, nachádza aj v reťazci *retazec2*. Teda aktuálnu hodnotu premennej *znak* vypíšeme na obrazovku pomocou:

**print***(znak)*

5) Výsledný program viď ďalší slajd.

# Operátor **in - príklad**

test.py - C:\Users\wilczo\AppData\Local\Programs\Python\Python312\test.py (3.12.6)

File Edit Format Run Options Window Help

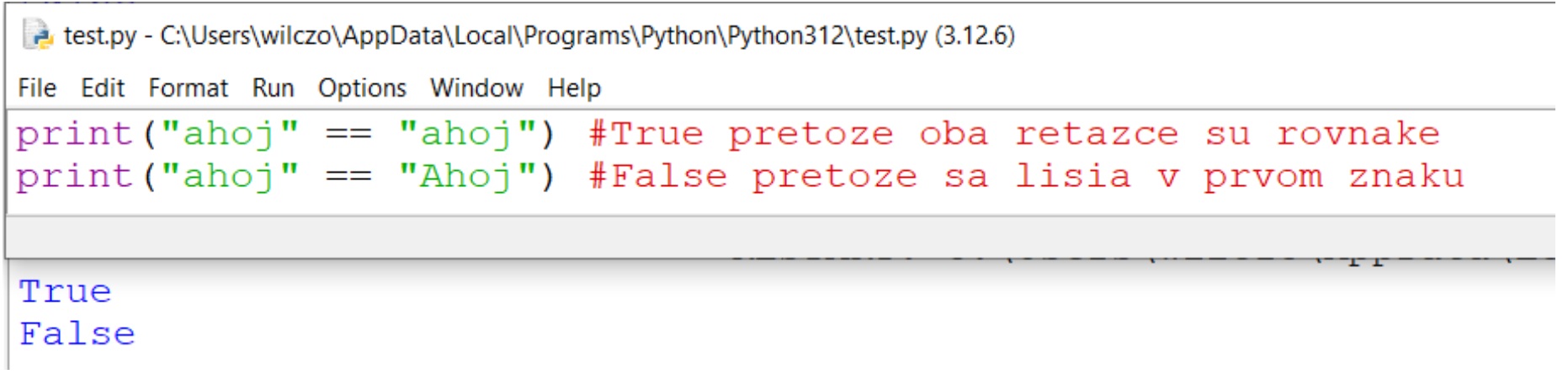
```
def v_oboch(retazec1, retazec2):
    for znak in retazec1:
        if (znak in retazec2) == True:
            print(znak)

v_oboch("aeiouy","abeceda") #vypise samohlasky (znaky z retazca "aeiouy")
                            #nachadzajuce sa v retazci "abeceda"
```

```
a
e
```

# Porovnávanie reťazcov

V jazyku Python dokonca môžeme reťazce aj porovnávať:

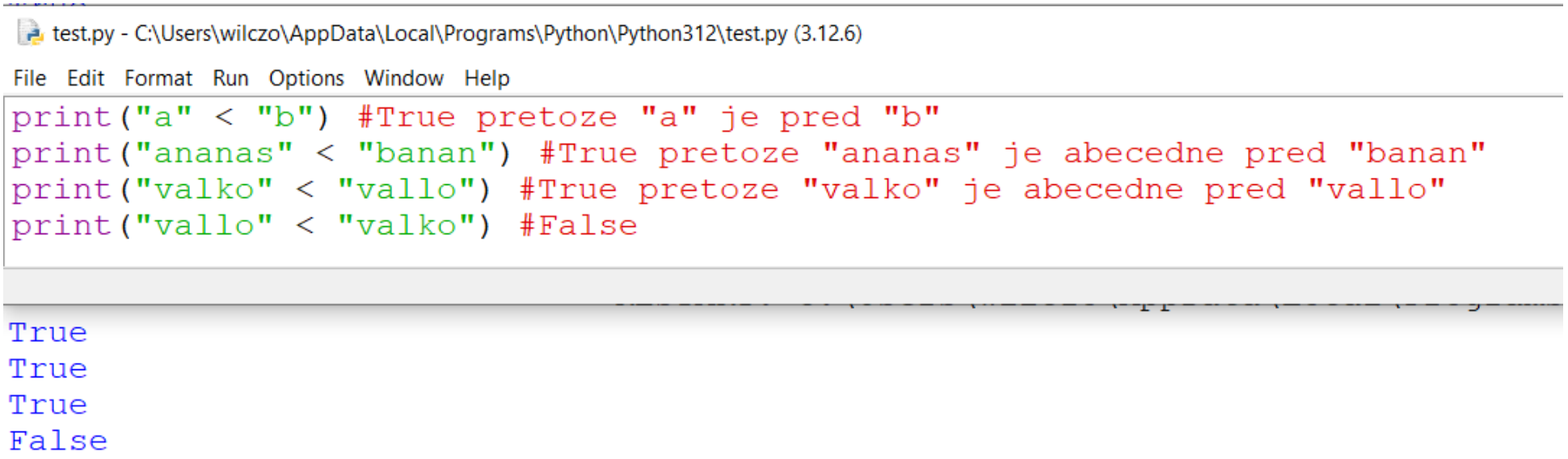
1) Vieme testovať, či sú 2 reťazce rovnaké alebo nie pomocou komparátorov == a !=



# Porovnávanie reťazcov

V jazyku Python dokonca môžeme reťazce aj porovnávať:

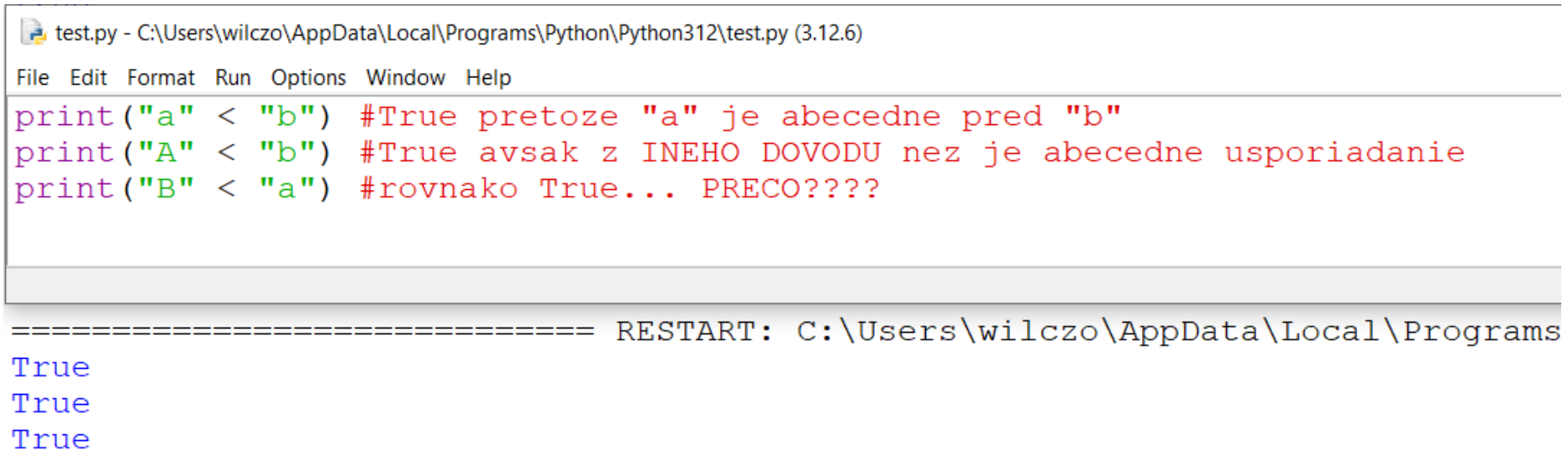
2) Vieme testovať, či sú 2 reťazce v abecednom (lexikografickom) usporiadaní



# Porovnávanie reťazcov

Testovanie na lexikografické (abecedné) usporiadanie však funguje korektne len ak majú oba reťazce ROVNAKÉ veľkosti písmen!!!

Všimnite si:



# Porovnávanie reťazcov

Porovnávanie reťazcov funguje na princípe reprezentácie dát v počítači.

Konkrétne pri reťazcoch platí, že **každý** textový znak je v počítači reprezentovaný ako nejaké **číslo**. To, aké číslo prináleží ktorému textovému znaku závisí na použitom tzv. *kódovaní*.

*Kódovanie* je spôsob reprezentácie (kódovania) dát v počítači, t.j. spôsob, ako reprezentujeme informácie pomocou núl a jednotiek. V prípade textu platí, že každý textový znak je nejakým ustanoveným spôsobom reprezentovaný nulami a jednotkami – teda nejakým konkrétnym číslom.

Pre bežné anglické znaky sa spravidla používa napríklad kódovanie pomocou tzv. ASCII tabuľky (American Standard Code for Information Interchange) – ide o tabuľku, ktorá bežným znakom (tlačiteľným – písmená, číslice, symboly, ale i netlačiteľným) prideľuje čísla od 0 po 127.

# ASCII tabuľka (vľavo číselný kód, vpravo symbol)

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NUL | 16 | DLE | 32 | SP | 48 | 0 | 64 | @ | 80 | P | 96 | ‘ | 112 | p |
| 1 | SOH | 17 | DC1 | 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 2 | STX | 18 | DC2 | 34 | " | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 3 | ETX | 19 | DC3 | 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| 4 | EOT | 20 | DC4 | 36 | $ | 52 | 4 | 68 | D | 84 | T | 100 | d | 116 | t |
| 5 | ENQ | 21 | NAK | 37 | % | 53 | 5 | 69 | E | 85 | U | 101 | e | 117 | u |
| 6 | ACK | 22 | SYN | 38 | & | 54 | 6 | 70 | F | 86 | V | 102 | f | 118 | v |
| 7 | BEL | 23 | ETB | 39 | ’ | 55 | 7 | 71 | G | 87 | W | 103 | g | 119 | w |
| 8 | BS | 24 | CAN | 40 | ( | 56 | 8 | 72 | H | 88 | X | 104 | h | 120 | x |
| 9 | HT | 25 | EM | 41 | ) | 57 | 9 | 73 | I | 89 | Y | 105 | i | 121 | y |
| 10 | LF | 26 | SUB | 42 | * | 58 | : | 74 | J | 90 | Z | 106 | j | 122 | z |
| 11 | VT | 27 | ESC | 43 | + | 59 | ; | 75 | K | 91 | [ | 107 | k | 123 | { |
| 12 | FF | 28 | FS | 44 | ’ | 60 | < | 76 | L | 92 | \ | 108 | l | 124 | \| |
| 13 | CR | 29 | GS | 45 | – | 61 | = | 77 | M | 93 | ] | 109 | m | 125 | } |
| 14 | SO | 30 | RS | 46 | · | 62 | > | 78 | N | 94 | ^ | 110 | n | 126 | ~ |
| 15 | SI | 31 | US | 47 | / | 63 | ? | 79 | O | 95 | _ | 111 | o | 127 | DEL |

# Porovnávanie reťazcov

Všimnite si, že v tabuľke sú malé písmená usporiadané za sebou, od “a“ po “z“, pričom ich číselný kód rastie.

Analogicky aj veľké písmená sú usporiadané od “A“ po “Z“ a ich číselný kód rastie. Avšak veľké písmená sú v ASCII tabuľke **pred** malými písmenami, teda číselné kódy veľkých písmen sú **menšie** než číselné kódy malých písmen.

Preto je v Pythone logický výraz (“B“ < “a“) vyhodnotený ako *True*, pretože znak “B“ má **menší číselný kód** než znak “a“.

Python teda lexikografické usporiadanie reťazcov odvádza na základe číselných kódov príslušných znakov.

# Vstavané funkcie **ord()** a **chr()**

V jazyku Python sú k dispozícii vstavané funkcie **ord()** a **chr()**, pomocou ktorých je možné konvertovať **znaky** na ich **číselné kódy** a naopak, **číselné kódy** na príslušné **znaky**.

Funkcia **ord***(znak)* **vráti** číselný kód zodpovedajúci znaku *znak*.

Funkcia **chr***(cislo)* **vráti** znak zodpovedajúci číselnému kódu *cislo.*

# Vstavané funkcie **ord()** a **chr()**

test.py - C:\Users\wilczo\AppData\Local\Programs\Python\Python312\test.py (3.12.6)

File Edit Format Run Options Window Help

```
#nasledovne 4 volania vypisu postupne ciselne kody znakov 'A', 'B', 'a', 'b'
print(ord('A'))
print(ord('B'))
print(ord('a'))
print(ord('b'))
#nasledovne 4 volania vypisu postupne znaky zodpovedajuce cislenym kodom 65, 66, 97, 98
print(chr(65))
print(chr(66))
print(chr(97))
print(chr(98))
```

```
65
66
97
98
A
B
a
b
```