Predavanje02 - Tuples, Dictionaries, Sets, Flow Control Statements

February 25, 2022

1 Tuples

Imajo enake lastnosti kot list, vendar so "immutable" - Njihovih vrednosti ne moremo spreminjati.

Definira se jih z navadnimi oklepaji ()

[1]:
```
t  = ("pingvin", "medved", "los", "volk")
print(t)
```

```
('pingvin', 'medved', 'los', 'volk')
```

[2]:
```
# Primer: Touples are ordered
t  = ("pingvin", "medved", "los", "volk")
t2  = ("pingvin", "volk", "medved", "los")

if t == t2:
    print("Touples are NOT ordered")
else:
    print("Touples ARE ordered")
```

```
Touples ARE ordered
```

[3]:
```
# Primer: Touples can contain any arbitrary object
t  = ("pingvin", "medved", "los", "volk", 1.23, True)
print(t)
```

```
('pingvin', 'medved', 'los', 'volk', 1.23, True)
```

[4]:
```
# Primer: Touples are indexed
t  = ("pingvin", "medved", "los", "volk")
print(t)

print(t[2]) # primer, da so elementi indexirani

print(t[1:3]) # slicing primer
```

```
('pingvin', 'medved', 'los', 'volk')
los
('medved', 'los')
```

[5]:
```
# Primer: Touples can be nested
t  = ("pingvin", "medved", "los", "volk", ("lisica", "krava"))
print(t)
```

```
('pingvin', 'medved', 'los', 'volk', ('lisica', 'krava'))
```

[6]:
```
# Primer: Touples are IMMUTABLE
t = ("pingvin", "medved", "los", "volk")
t[1] = "Bork!"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-1f7dd710d595> in <module>
      1 # Primer: Touples are IMMUTABLE
      2 t = ("pingvin", "medved", "los", "volk")
----> 3 t[1] = "Bork!"

TypeError: 'tuple' object does not support item assignment
```

[7]:
```
# Primer: da je dinamična
t  = ("pingvin", "medved", "los", "volk")
print(t)
print(type(t))

t = 2
print(t)
print(type(t))
```

```
('pingvin', 'medved', 'los', 'volk')
<class 'tuple'>
2
<class 'int'>
```

Zakaj bi uporabljali touple namesto list?

- Program je hitrejši, če manipulira z touple kot pa z list
- Če ne želimo spreminjati elementov

TECHNICAL

Treba pazit kadar inicializiramo touple samo z eno vrednostjo.

[8]:
```
t = (1,2,3,4) # nebi smel bit problem
print(t)
print(type(t))
print()

t = () # nebi smel bit problem. Prazen touple
print(t)
```

```
print(type(t))
print()

t = (2) # kle nastane problem
print(t)
print(type(t))
print()
'''
Since parentheses are also used to define operator precedence in expressions,␣
 ↪Python evaluates
the expression (2) as simply the integer 2 and creates an int object. To tell␣
 ↪Python that you really want
to define a singleton tuple, include a trailing comma (,) just before the␣
 ↪closing parenthesis.
'''

t = (2,)
print(t)
print(type(t))
```

```
(1, 2, 3, 4)
<class 'tuple'>

()
<class 'tuple'>

2
<class 'int'>

(2,)
<class 'tuple'>
```

[ ]:

1.1 Vaja

Naloga: Iz sledečega tuple pridobite vrednost cc

```
our_tuple = ("a", ["bb", "cc"], "d", [("eee"), ["ffff"], "ggg"])
```

[9]:
```
# Rešitev
our_tuple = ("a", ["bb", "cc"], "d", [("eee"), ["ffff"], "ggg"])

print(our_tuple[1][1])
```

```
cc
```

1.2 Vaja

Naloga: Pri sledečem tuple vzemite zadnjih 5 vrednosti.

```
our_tuple = (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)

Rešitev:
(16, 17, 18, 19, 20)
```

[10]:
```
# Rešitev
our_tuple = (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)
print(our_tuple[-5:])
```

```
(16, 17, 18, 19, 20)
```

[ ]:
```

```

2 Dictionaries

Njihove lastnosti so sledeče: * Are insertion ordered (vrstni red elementov je odvisen od vrstega reda dodajanja) (to velja od python 3.6+) * Element accession (do elementov se dostopa preko ključev, ne preko indexov) * Can be nested (kot element ima lahko še en dictionary, list, touple, ...) * Are mutable (vrednosti elementov se lahko spreminjajo) * Are dynamic (sej to velja za vse pr pythonu)

Dictionary je sestavljen iz parov ključa in vrednosti. Vsak Ključ ima svojo vrednost.

[11]:
```
d = {
'macek' : 'Silvestre',
'pes'   : 'Fido',
'papagaj': 'Kakadu'
}
print(d)
print(type(d))
```

```
{'macek': 'Silvestre', 'pes': 'Fido', 'papagaj': 'Kakadu'}
<class 'dict'>
```

[12]:
```
# Primer: Can contain any arbitrary objects
d = {
'macek' : 1,
'pes'   : 'Fido',
'papagaj': False
}
print(d)
```

```
{'macek': 1, 'pes': 'Fido', 'papagaj': False}
```

2.0.1 Accessing dictionary value

Vrednosti najdemo preko ključev.

[13]:
```
d = {
'macek' : 'Silvestre',
'pes'   : 'Fido',
'papagaj': 'Kakadu'
}
print(d['papagaj'])
```

```
Kakadu
```

Če vpišemo ključ, ki ne obstaja python vrne napako.

[14]:
```
d = {
'macek' : 'Silvestre',
'pes'   : 'Fido',
'papagaj': 'Kakadu'
}
d['koza'] # should give KeyError
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-14-91bc34ee01c4> in <module>
      4 'papagaj': 'Kakadu'
      5 }
----> 6 d['koza'] # should give KeyError

KeyError: 'koza'
```

Dodajanje novih vrednosti

[15]:
```
d = {
'macek' : 'Silvestre',
'pes'   : 'Fido',
'papagaj': 'Kakadu'
}
d['koza'] = "Micka"
print(d)
```

```
{'macek': 'Silvestre', 'pes': 'Fido', 'papagaj': 'Kakadu', 'koza': 'Micka'}
```

Posodabljanje vrednosti.

[16]:
```
d['koza'] = 'Helga'
print(d)
```

```
{'macek': 'Silvestre', 'pes': 'Fido', 'papagaj': 'Kakadu', 'koza': 'Helga'}
```

Brisanje elementa.

[17]:
```
del d['koza']
print(d)
```

```
{'macek': 'Silvestre', 'pes': 'Fido', 'papagaj': 'Kakadu'}
```

3 Restrictions on dictionary keys

Kot Ključ lahko uporabimo poljubne vrednosti, dokler so "immutable". Sm spadajo integer, float, string, boolean, touple.

Touple je lahko ključ le, če so elementi znotraj njega tudi "immutable" (strings, integers, floats,...).

[18]:
```
d = {1: 'a',
     2.3: 'b',
     "string": 'c',
     None: 'd',
     (1,"touple"):'e',
    }
print(d)
print(d[2.3])
print(d[None])
print(d[(1, "touple")])
# PAZI: Če daš True namest None bo narobe deloval. Pomojm tretira 1 kt True pa␣
 ↪se mu zmeša mal.
# Sej keywords dajat sm je nesmiselno
```

```
{1: 'a', 2.3: 'b', 'string': 'c', None: 'd', (1, 'touple'): 'e'}
b
d
e
```

[19]:
```
# Primer: Vrže error, ker hočemo kot ključ uporabiti list, ki pa je mutable
d = {[1,1]: 'a', [1,2]: 'b'}
d = {(1,2,[1,2]): "f",}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-f98311767e81> in <module>
      1 # Primer: Vrže error, ker hočemo kot ključ uporabiti list, ki pa je␣
 ↪mutable
----> 2 d = {[1,1]: 'a', [1,2]: 'b'}
      3 d = {(1,2,[1,2]): "f",}

TypeError: unhashable type: 'list'
```

4 Technical Note:

Why does the error message say "unhashable" rather than "mutable"? Python uses hash values internally to implement dictionary keys, so an object must be hashable to be used as a key.

https://docs.python.org/3/glossary.html#term-hashable

> An object is hashable if it has a hash value which never changes during its lifetime (it needs a **hash**() method), and can be compared to other objects (it needs an **eq**() method). Hashable objects which compare equal must have the same hash value.
>
> Hashability makes an object usable as a dictionary key and a set member, because these data structures use the hash value internally.
>
> All of Python's immutable built-in objects are hashable; mutable containers (such as lists or dictionaries) are not. Objects which are instances of user-defined classes are hashable by default. They all compare unequal (except with themselves), and their hash value is derived from their id().

AMPAK

Ključ more bit edinstven (se ne sme ponovit):

[20]:
```
d = {
'macek' : 'Silvestre',
'pes'   : 'Fido',
'papagaj': 'Kakadu',
'macek' : 'Amadeus'
}
print(d)
```

```
{'macek': 'Amadeus', 'pes': 'Fido', 'papagaj': 'Kakadu'}
```

5 Built-in Dictionary Methods

Še nekaj ostalih metod.

`d.clear()`

d.clear() empties dictionary d of all key-value pairs:

[21]:
```
d = {'a': 10, 'b': 20, 'c': 30}
print(d)

d.clear()
print(d)
```

```
{'a': 10, 'b': 20, 'c': 30}
{}
```

`d.get(<key>[, <default>])`

get() metoda nam nudi preprost način kako dobimo vrednost ključa brez, da preverimo, če ključ sploh obstaja.

Če ključ ne obstaja dobimo None

[22]:
```
d = {'a': 10, 'b': 20, 'c': 30}
print(d.get('b'))
print(d.get('z'))
```

```
20
None
```

Če ključ ni najden in smo specificirali dodaten argument nam vrne le tega namesto None.

[23]:
```
d = {'a': 10, 'b': 20, 'c': 30}
print(d.get('z', -5))
```

```
-5
```

`d.items()`

Vrne nam list sestavljen iz touple, ki so sestavljeni iz ključ-vrednost parov. Prvi element toupla je ključ, drugi je vrednost.

[24]:
```
d = {'a': 10, 'b': 20, 'c': 30}
print(list(d.items()))
print(list(d.items())[1])
print(list(d.items())[1][1])
```

```
[('a', 10), ('b', 20), ('c', 30)]
('b', 20)
20
```

`d.keys()`

Vrne nam list ključev.

[25]:
```
d = {'a': 10, 'b': 20, 'c': 30}
print(list(d.keys()))
```

```
['a', 'b', 'c']
```

`d.values()`

Vrne nam list vrednosti.

[26]:
```
d = {'a': 10, 'b': 10, 'c': 10}
print(list(d.values()))
```

```
[10, 10, 10]
```

`d.pop(<key>[, <default>])`

Če ključ obstaja v dictionary ga odstrani skupaj z njegovo vrednostjo.

[27]:
```
d = {'a': 10, 'b': 20, 'c': 30}
print(d.pop('b'))
print(d)
```

```
20
{'a': 10, 'c': 30}
```

Če ne najde ključa nam vrne napako.

[28]:
```
d = {'a': 10, 'b': 20, 'c': 30}
print(d.pop('z'))
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-28-f36d736a326b> in <module>
      1 d = {'a': 10, 'b': 20, 'c': 30}
----> 2 print(d.pop('z'))

KeyError: 'z'
```

Če ključ ni najden, smo pa dodatno specificirali default argument, potem nam vrne vrednost default argumenta in ne dvigne nobene napake.

[29]:
```
d = {'a': 10, 'b': 20, 'c': 30}
print(d.pop('z', "Ni našlo ključa"))
```

```
Ni našlo ključa
```

`d.popitem()`

Odstrani random, arbitrarni ključ-vrednost par in nam ga vrne kot touple. >popitem() is useful to destructively iterate over a dictionary, as often used in set algorithms

[30]:
```
d = {'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': 50, 'f': 60, 'g': 70}
print(d.popitem())
print(d)
```

```
('g', 70)
{'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': 50, 'f': 60}
```

Če je dictionary prazen dobimo KeyError error.

[31]:
```
d = {}
d.popitem()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-31-3d5a99fd0340> in <module>
      1 d = {}
----> 2 d.popitem()

KeyError: 'popitem(): dictionary is empty'
```

[32]:
```
# Primer: Da je dictionary dinamičen
d = {'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': 50, 'f': 60, 'g': 70}
print(d)
print(type(d))

d = 1.2
print(d)
print(type(d))
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': 50, 'f': 60, 'g': 70}
<class 'dict'>
1.2
<class 'float'>
```

[ ]:
```
```

[33]:
```
dict_ = {
    "a": 1,
    "b": 2,
    "c": 3,
    "d": 4
}
```

5.1 Vaja 03

Naloga: Sledečemu dictionary zamenjanjte vrednost pod ključem b v vrednost 12 in odstranite vrednost pod ključem d.

```
our_dict = {
    "a": 10,
    "b": 9,
    "c": 8,
    "d": 7,
    "e": 3
}
```

[34]:
```
our_dict = {
    "a": 10,
    "b": 9,
    "c": 8,
    "d": 7,
    "e": 3
}

our_dict["b"] = 12
del our_dict["d"]

print(our_dict)
```

```
{'a': 10, 'b': 12, 'c': 8, 'e': 3}
```

5.2 Vaja 04

Naloga: Iz sledečega dictionary pridobite vrednost fff.

```
d = {
    "a": "a",
    "b": "b",
    "c": {
        1: 11,
        2: 22,
        3: 33,
        4: {
            5: "ccc",
            6: "ddd",
            "7": "fff"
        }
    }
}
```

[35]:
```
d = {
    "a": "a",
    "b": "b",
    "c": {
        1: 11,
        2: 22,
        3: 33,
        4: {
            5: "ccc",
            6: "ddd",
            "7": "fff"
        }
    }
}

print(d["c"][4]["7"])
```

```
fff
```

6 Sets

(Množice)

Ta data-tip je prav tako kolekcija elementov, ampak nad set-i se da izvajati posebne operacije.

Lastnosti * Sets are unordered * Set elements are unique. Duplicate elements are not allowed * A set itself may be modified, but the elements must be of type immutable.

- Sets do not support indexing, slicing, or other sequence-like behavior. (Če hočš pridt do specifičnega elementa lahko uporabiš for _ in set:)

[36]:
```
s = {"medved", "zajec", "volk", "slon", "zajec"} # zajec se ne ponovi ampak je␣
↪samo 1x izpisan!
print(s)
print(type(s))
```

```
{'medved', 'volk', 'slon', 'zajec'}
<class 'set'>
```

Set je lahko prazen ampak Python bo {} prebral kot prazen dictionary.

Edini način, da ustvarimo prazen set je z set().

[37]:
```
s = {}
print(type(s))

s = set()
print(type(s))
```

```
<class 'dict'>
<class 'set'>
```

Elementi so lahko poljubni ampak morajo biti "immutable."

[38]:
```
s = {42, 'eee', (1, 2, 3), 3.14159} # touple je lahko element, ker je immutable
print(s)
```

```
{'eee', 42, 3.14159, (1, 2, 3)}
```

[39]:
```
s = {11, [1, 2, 3], 'eeee'} # list ne more bit element, ker je mutable
print(x)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-d32f290234c5> in <module>
----> 1 s = {11, [1, 2, 3], 'eeee'} # list ne more bit element, ker je mutable
      2 print(x)

TypeError: unhashable type: 'list'
```

[40]:
```
# Primer: Can be nested
s = {42, 'eee', (1, 2, (4, 5, 6)), 3.14159}
print(s)
```

```
{'eee', 42, 3.14159, (1, 2, (4, 5, 6))}
```

6.1 Operating on a Set

Nad set-i je možno izvajanje posebnih operacij.

Večina jih je lahko zapisana na dva načina: z metodo ali z operatorjem.

Union

Vsebuje elemente iz obeh set-ov.

[41]:
```
x1 = {1, 2, 3, 4}
x2 = {4, 5, 6, 7}
print(x1 | x2) # operator
print(x1.union(x2)) # metoda
# Element 4 je samo enkrat, ker se elementi v set-ih ne ponavljajo
```

```
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 3, 4, 5, 6, 7}
```

Razlika med operatorjem in metodo je, da operator zahteva, da sta obe spremenljivki set, medtem ko metoda uzeme kot argument poljuben "iterable".

[42]:
```
print(x1.union(list(x2)))
print(x1 | list(x2)) # this one should throw error
```

```
{1, 2, 3, 4, 5, 6, 7}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-42-ea3a52617f77> in <module>
      1 print(x1.union(list(x2)))
----> 2 print(x1 | list(x2)) # this one should throw error

TypeError: unsupported operand type(s) for |: 'set' and 'list'
```

Ali z operatorjem ali z metodo lahko specificiramo večje število set-ov.

[43]:
```
x1 = {1, 2, 3, 4}
x2 = {4, 5}
x3 = {5, 6}
x4 = {9, 10}

print(x1.union(x2, x3, x4))

print(x1 | x2 | x3 | x4)
```

```
{1, 2, 3, 4, 5, 6, 9, 10}
{1, 2, 3, 4, 5, 6, 9, 10}
```

Intersection

Vrne set z elementi skupni obema set-oma.

[44]:
```
x1 = {1, 2, 3, 4}
x2 = {4, 5, 6, 7}
```

```
print(x1.intersection(x2))
print(x1 & x2)
```

```
{4}
{4}
```

Difference

Vrne set z elementi, ki so v x1 in ne v x2.

[45]:
```
x1 = {1, 2, 3, 4}
x2 = {4, 5, 6, 7}

print(x1.difference(x2))
print(x1 - x2)
```

```
{1, 2, 3}
{1, 2, 3}
```

Symetric Difference

Vrne set z elementi, ki so ali v prvem ali v drugem set-u, vendar ne v obeh.

[46]:
```
x1 = {1, 2, 3, 4}
x2 = {4, 5, 6, 7}

print(x1.symmetric_difference(x2))
print(x1 ^ x2)
```

```
{1, 2, 3, 5, 6, 7}
{1, 2, 3, 5, 6, 7}
```

še več teh metod https://realpython.com/python-sets/

7 Modifying Sets

Set-e lahko tudi spreminjamo.

`x.add(<elem>) adds <elem>, which must be a single immutable object, to x:`

[47]:
```
x = {1, 2, 3, 4}
x.add("string")
print(x)
```

```
{1, 2, 3, 4, 'string'}
```

`x.remove(<elem>) removes <elem> from x. Python raises an exception if <elem> is not in x:`

[48]:
```
x = {1, 2, 3, 4}
print(x.remove(3))
print(x)
```

```
None
{1, 2, 4}
```

[49]:
```
x = {1, 2, 3, 4}
x.remove(6) # gives error
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-49-4f450e54381b> in <module>
      1 x = {1, 2, 3, 4}
----> 2 x.remove(6) # gives error

KeyError: 6
```

x.discard(<elem>) also removes <elem> from x. However, if <elem> is not in x, this method quie

[50]:
```
x = {1, 2, 3, 4}
print(x.discard(2))
print(x)
```

```
None
{1, 3, 4}
```

[51]:
```
x = {1, 2, 3, 4}
x.discard(6) # doesnt give error
print(x)
```

```
{1, 2, 3, 4}
```

python x.pop() removes and returns an arbitrarily chosen element from x. If x is empty, x.pop() raises an exception:

[52]:
```
x = {1, 2, 3, 4}
print("First pop: ", x.pop())
print(x)

print("Second pop: ",x.pop())
print(x)

print("Third pop: ", x.pop())
print(x)

print("Fourth pop: ",x.pop())
print(x)

print("Fourth pop: ",x.pop()) #this one should give error
print(x)
```

```
First pop:  1
```

```
{2, 3, 4}
Second pop:  2
{3, 4}
Third pop:  3
{4}
Fourth pop:  4
set()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-52-594f9b06ae7d> in <module>
     12 print(x)
     13
---> 14 print("Fourth pop: ",x.pop()) #this one should give error
     15 print(x)

KeyError: 'pop from an empty set'
```

x.clear() removes all elements from x

[53]:
```
x = {1, 2, 3, 4}
print(x)
x.clear()
print(x)
```

```
{1, 2, 3, 4}
set()
```

7.1 Frozen sets

Python ima tudi frozenset, ki se obnaša isto kot set, le da je frozenset immutable.

[54]:
```
x = frozenset([1, 2, 3, 4])
print(x | {9, 8, 7}) # you can perform non-modifying operations on frozensets
x.add(100) # should give error, because frozenset is immutable
```

```
frozenset({1, 2, 3, 4, 7, 8, 9})
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-54-769f87059d2e> in <module>
      1 x = frozenset([1, 2, 3, 4])
      2 print(x | {9, 8, 7}) # you can perform non-modifying operations on␣
 ↪frozensets
----> 3 x.add(100) # should give error, because frozenset is immutable

AttributeError: 'frozenset' object has no attribute 'add'
```

7.2 Vaja 05

Naloga: Iz sledečega lista odstranite vrednost, ki se nahaja na indexu 4. Vrednost dodajte v dictionary pod ključ d.

Nato iz dictionary pridobite vse vrednosti. Te vrednosti shranite v nov set in novonastali set primerjajte ali je enak podanemu set-u.

```
our_list = [1,2,3,4,5,6,7]
our_dict = {
    "a": 2,
    "b": 5,
    "c": 8,
    "d": 12,
    "e": 357,
    "f": 12
}
our_set = {357, 12, 12, 8, 5, 2, 2}
```

[55]:
```
our_list = [1,2,3,4,5,6,7]
our_dict = {
    "a": 2,
    "b": 5,
    "c": 8,
    "d": 12,
    "e": 357,
    "f": 12
}
our_set = {357, 12, 12, 8, 5, 2, 2}


x = our_list.pop(4)
print(x)

our_dict["d"] = x
print(our_dict)
print(our_dict.values())

x_set = set(our_dict.values())
x_set == our_set
```

```
5
{'a': 2, 'b': 5, 'c': 8, 'd': 5, 'e': 357, 'f': 12}
dict_values([2, 5, 8, 5, 357, 12])
```

[55]:
```
True
```

[ ]:
```

```

7.3 Vaja 06

Naloga: Izpišite vse skupne črke sledečih dveh stringov.

```
str1 = "Danes je lep dan"
str2 = "Jutri bo deževalo"

OUTPUT:
{' ', 'a', 'd', 'e', 'l'}
```

[56]:
```
str1 = "Danes je lep dan"
str2 = "Jutri bo deževalo"

set1 = set(str1)
set2 = set(str2)


set1.intersection(set2)
```

[56]:
```
{' ', 'a', 'd', 'e', 'l'}
```

8 FLOW CONTROL STATEMENTS

Omogočajo nam kontrolo sprememb in logike programa.

8.1 If statement

```
if <expr>:
    <statement>
    <statement>
    ...
    <statement>
<following_statement>


<expr> je izraz ovrednoten v Boolean kontekstu.
<statement> je Python izraz (nadaljevanje naše kode), ki je pravilno zamaknjen.
Če je <expr> True, potem se izvedejo <statement>. Če je <expr> False, potem se  <statement> pr
Nato se program nadaljuje z <following_statement>
```

Indentation / Zamikanje

> Pri Pythonu se zamikanje (indentation) uporablja za definiranje blokov kode. Vse vrstice z istim zamikom se smatrajo kot isti blok kode.
>
> Bloke kode se lahko poljubno globoko "nesta".
>
> Zamikanje je določeno z tabulatorjem ali presledki. Ni važno točno število, važno je, da je skozi kodo enako.

[57]:
```
x = 0
y = 5

if x < y:
    print("Smo znotraj if.")
    print("End if")
print("End")
```

```
Smo znotraj if.
End if
End
```

8.2 Else

Včasih želimo, da če je nekaj res se izvede določen blok kode, če stvar ni res pa naj se izvede drug del kode.

To dosežemo z else.

```
if <expr>:
    <statement(s)>
else:
    <statement(s)>
<following_statement>
```

Če je <expr> True se izvede blok direktno pod njem, če pa je <expr> False se ta blok kode presk

[58]:
```
x = 100

if x < 50:
    print('(first block)')
    print('x is small')
else:
    print('(second block)')
    print('x is large')

print("End")
```

```
(second block)
x is large
End
```

8.3 Elif

Če želimo še večjo razvejanost naših možnosti lahko uporabimo elif (else if).

```
if <expr>:
    <statement(s)>
elif <expr>:
    <statement(s)>
```

```
elif <expr>:
    <statement(s)>
else:
    <statement(s)>
<following_statement>
```

Python preveri vsak <expr> posebej. Pri ta prvem, ki bo True, bo izvedel njegov blok kode. Če ni nobeden True se bo izvedel else blok kode.

[59]:
```
x = 20
if x > 100:
    print('x je večje od 100')
elif x > 50:
    print('x večje od 50 in manjše od 100')
elif x > 30:
    print('x večje od 30 in manjše od 50')
elif x > 10:
    print('x večje od 10 in manjše od 30')
else:
    print("x manjše od 10")

print("End")
```

```
x večje od 10 in manjše od 30
End
```

8.4 One-line if statement

Obstaja način zapisa if stavka v eni vrstici ampak se ta način odsvetuje, ker napravi kodo nepregledno.

https://realpython.com/python-conditional-statements/

```
<expr1> if <conditional_expr> else <expr2>

z = 1 + x if x > y else y + 2

If <conditional_expr> is true, <expr1> is returned and <expr2> is not evaluated.
If <conditional_expr> is false, <expr2> is returned and <expr1> is not evaluated.
```

[60]:
```
x = 8
z = 1 + x if x > 10 else x**2
print(z)

x = 20
z = 1 + x if x > 10 else x**2
print(z)
```

```
64
21
```

8.5 The pass statements

Uporablja se kot “placeholder”, da nam interpreter ne meče napak.

[61]:
```
if True:

print("Hello") # should give IndentationError
```

```
  File "<ipython-input-61-33a91c099307>", line 3
    print("Hello") # should give IndentationError
    ^
IndentationError: expected an indented block
```

[62]:
```
if True:
    pass
print("Hello") # should be fine now with the pass added
```

```
Hello
```

[ ]:
```

```

[ ]:
```

```