Vislice

Naslednji program katerega bomo napisali je igra **Vislice**.

Pravila igre so sledeča:

> Igralec zmaga, če uspe uganiti iskano besedo. Ugiba lahko po eno črko naenkrat. Če je ugibana črka pravilan, se črko dopiše kjerkoli se pojavi v besedi. Če je ugibana črka napačna, potem je igralec "izgubil en ugib". Na voljo ima 10 ugibov. Če izgubi vse ugibe, potem igralec izgubi igro.

Primer igranja igre:

```
Guesses so far [].
What is your guess? a
_ a_ _ _ _ _

Guesses so far ['a'].
What is your guess? e
_ a_ _ _ _ _

Guesses so far ['a', 'e'].
What is your guess? o
_ a_ o_ _ o

Guesses so far ['a', 'e', 'o'].
What is your guess? p
_ a_ o_ _ o

Guesses so far ['a', 'e', 'o', 'p'].
What is your guess? r
_ a_ o_ _ o

Guesses so far ['a', 'e', 'o', 'p', 'r'].
What is your guess? l
_ a_ ol_ o

Guesses so far ['a', 'e', 'o', 'p', 'r', 'l'].
What is your guess? k
_ a_ olko

Guesses so far ['a', 'e', 'o', 'p', 'r', 'l', 'k'].
What is your guess? j
ja_ olko

Guesses so far ['a', 'e', 'o', 'p', 'r', 'l', 'k', 'j'].
What is your guess? b
jabolko
KONEC
True
```

Naš program bi okvirno izgledal sledeče:

```
# Ustvarimo spremenljivko v katero shranimo besedo katero iščemo

# 1. Izpišemo _ _ _ _ _ _ (in dodamo katere črke so bile uganjene)

# 2. Uporabnika uprašamo naj vtipka katero črko ugiba

# 3. Preverimo ali je črka del besede

# 4. Preverimo ali je uganil besedo:

# 4.a   Če je uganil besedo je zmagal. Konec igre

# 4.b   Če ni uganil besede in nima več možnih ugibov je izgubil. Konec igre

# 4.c   Če ni uganil besede in ima še ugibe gremo na korak 1.
```

Ustvarimo **vislice.py** datoteko, v katero bomo pisali naš program.

Zaenkrat bo beseda, katero igralec ugiba, *hard-coded* v našem programu.

Začeli bomo tako, da od igralca zahtevamo naj vnese črko, katero ugiba.

Vislice

Naloga:

Ustvarite spremenljivko v kateri bomo hranili besedo, katero mora igralec uganiti. Izpišite to besedo.

Od uporabnika zahtevajte naj vnese črko in to črko izpišite.

```
Iskana beseda je: lokomotiva
Vnesi ugibano črko: c
Ugibalo se je črko c
```

In [1]:
```
# Ustvarimo spremenljivko v katero shranimo besedo katero iščemo
beseda = "lokomotiva"
print(f"Iskana beseda je: {beseda}")

# 1. Izpišemo _ _ _ _ _ _ (in dodamo katere črke so bile uganjene)

# 2. Uporabnika uprašamo naj vtipka katero črko ugiba
ugib = input("Vnesi ugibano črko: ")
print(f"Ugibalo se je črko {ugib}")

# 3. Preverimo ali je črka del besede

# Loading [MathJax]/extensions/Safe.js ali je uganil besedo:
```

```
# 4.a   Če je uganil besedo je zmagal. Konec igre

# 4.b   Če ni uganil besede in nima več možnih ugibov je izgubil

# 4.c   Če ni uganil besede in ima še ugibe gremo na korak 1.
```

```
Iskana beseda je: lokomotiva
Vnesi ugibano črko: s
Ugibalo se je črko s
```

Sedaj preverimo ali se ugibana črka nahaja znotraj besede.

To lahko dosežemo z primerjalno operacijo `in`. Ta nam vrne `True`, če se naša vrednost nahaja v nekem `iterable`. V našem primeru lahko preverimo ali se črka nahaja znotraj besede.

In [6]:
```
ele = "a"
expr = ele in "abeceda"
print(type(expr), expr)
```

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

Vislice

Naloga:
Preverite ali se ugibana črka nahaja znotraj besede.

```
Če se črka nahaja znotraj besede izpišite `Pravilna črka`.
Če se črka NE nahaja znotraj besede izpišite `Nepravilna črka.`
```

In [2]:
```
# Ustvarimo spremenljivko v katero shranimo besedo katero iščemo
beseda = "lokomotiva"
print(f"Iskana beseda je: {beseda}")

# 1. Izpišemo _ _ _ _ _ _ (in dodamo katere črke so bile uganjene)

# 2. Uporabnika uprašamo naj vtipka katero črko ugiba
ugib = input("Vnesi ugibano črko: ")
print(f"Ugibalo se je črko {ugib}")

# 3. Preverimo ali je črka del besede
if ugib in beseda:
    print("Pravilna črka")
else:
    print("Nepravilna črka")

# 4. Preverimo ali je uganil besedo:

# 4.a   Če je uganil besedo je zmagal. Konec igre

# 4.b   Če ni uganil besede in nima več možnih ugibov je izgubil
```

```
# 4.c   Če ni uganil besede in ima še ugibe gremo na korak 1.
```

```
Iskana beseda je: lokomotiva
Vnesi ugibano črko: s
Ugibalo se je črko s
Nepravilna črka
```

Ker ima igralec maximalno 10 ugibov bomo 2. in 3. korak 10x ponovili.

Da določeno kodo večkrat ponovimo uporabimo **loops** (zanke).

While

While zanka se izvaja dokler se ne zgodi določen pogoj.

```
while <expr>:
    <while-loop statement(s)>

<following_statements>
```

Dokler je `<expr>` enak `True`, se bo izvajal **while blok kode**. Ko je `<expr>` enak `False` se nato nadaljuje z izvajanjem programa.

Vizualizacija kode

In [3]:
```
lepo_vreme = True
while lepo_vreme:
    print('Vreme je lepo.')
    lepo_vreme = False

print("Nadaljevanje programa")
```

```
Vreme je lepo.
Nadaljevanje programa
```

Vizualizacija kode - Vizualizacija neskončne zanke

In [4]:
```
#the body should be able to change the condition's value, because if the conditi

#lepo_vreme = True
#while lepo_vreme:
#    print('Vreme je lepo.')
#print("Nadaljevanje programa")

#ustavimo v CTRL + C
```

While zanko se lahko uporabi za ponovitev bloka kode določenega števila korakov.

Vizualizacija kode

In [5]:
```
i = 0
while i < 5:
    print(f'Repeated {i} times')
    i += 1

print("Nadaljevanje programa")
```

```
Repeated 0 times
Repeated 1 times
Repeated 2 times
Repeated 3 times
Repeated 4 times
Nadaljevanje programa
```

In [6]:
```
#A common use of the while loop is to do things like these:

temperature = 15

while temperature < 20:
    print('Heating...')
    temperature += 1

#Only instead of the temperature increasing continuously, we would e.g. get it f
#Remember to always have a way of exiting the loop! Otherwise it will run endles
```

```
Heating...
Heating...
Heating...
Heating...
Heating...
```

Vaja

Naloga:
Napišite program, ki izpiše prvih 10 sodih števil.

Vizualizacija kode

In [13]:
```
counter = 0
number = 1

while counter<10:
    if number%2 == 0:
        print(number)
        counter += 1
    number += 1
```

```
2
4
6
8
10
12
14
16
18
20
```

Vislice

Vizualizacija kode

Naloga:
Ustvarite novo spremenljivko, ki nam pove kolikokrat lahko igralec maximalno ugiba. V našem primeru ima igralec na voljo 10 poizkusov.

Nato s pomočjo **while loop-a** 10x od igralca zahtevajte naj ugiba in za vsak ugib preverite ali se črka nahaja znotraj iskane besede ali ne.

```
Output:

Iskana beseda je: lokomotiva

Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

Vnesi ugibano črko: b
Ugibalo se je črko b
Nepravilna črka

Vnesi ugibano črko: c
Ugibalo se je črko c
Nepravilna črka

Vnesi ugibano črko: d
Ugibalo se je črko d
Nepravilna črka

Vnesi ugibano črko: e
Ugibalo se je črko e
Nepravilna črka

Vnesi ugibano črko: f
Ugibalo se je črko f
Nepravilna črka

Vnesi ugibano črko: g
```

```
Ugibalo se je črko g
Nepravilna črka

Vnesi ugibano črko: h
Ugibalo se je črko h
Nepravilna črka

Vnesi ugibano črko: i
Ugibalo se je črko i
Pravilna črka

Vnesi ugibano črko: j
Ugibalo se je črko j
Nepravilna črka
```

In [16]:
```python
# Ustvarimo spremenljivko v katero shranimo besedo katero iščemo
beseda = "lokomotiva"
print(f"Iskana beseda je: {beseda}")

st_ugibov = 10

counter = 0
while counter < st_ugibov:
    # 1. Izpišemo _ _ _ _ _ _ (in dodamo katere črke so bile uganjene)
    print()

    # 2. Uporabnika uprašamo naj vtipka katero črko ugiba
    ugib = input("Vnesi ugibano črko: ")
    print(f"Ugibalo se je črko {ugib}")

    # 3. Preverimo ali je črka del besede
    if ugib in beseda:
        print("Pravilna črka")
    else:
        print("Nepravilna črka")

    # 4. Preverimo ali je uganil besedo:

    # 4.a   Če je uganil besedo je zmagal. Konec igre

    # 4.b   Če ni uganil besede in nima več možnih ugibov je izgubil

    # 4.c   Če ni uganil besede in ima še ugibe gremo na korak 1.

    counter += 1
```

```
Iskana beseda je: lokomotiva

Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

Vnesi ugibano črko: b
Ugibalo se je črko b
Nepravilna črka

Vnesi ugibano črko: c
Ugibalo se je črko c
Nepravilna črka

Vnesi ugibano črko: d
Ugibalo se je črko d
Nepravilna črka

Vnesi ugibano črko: e
Ugibalo se je črko e
Nepravilna črka

Vnesi ugibano črko: f
Ugibalo se je črko f
Nepravilna črka

Vnesi ugibano črko: g
Ugibalo se je črko g
Nepravilna črka

Vnesi ugibano črko: h
Ugibalo se je črko h
Nepravilna črka

Vnesi ugibano črko: i
Ugibalo se je črko i
Pravilna črka

Vnesi ugibano črko: j
Ugibalo se je črko j
Nepravilna črka
```

Da igralec ne bo ponesreči večkrat ugibal iste črke, mu bomo vsakič izpisali katere črke je že ugibal.

To pomeni, da moramo njegove ugibane črke nekam shranjevati.

To lahko dosežemo tako, da ustvarimo 10 spremenljivk in v vsako shranimo eno ugibano črko:

Naloga:
Ustvarite 10 spremenljivk, za vsak ugib eno. Vsakič, ko uporabnik ugiba shranite njegov ugib v novo spremenljivko.

ite vse ugibe katere je igralec že naredil.

Vizualizacija kode

In [8]:
```
# Ustvarimo spremenljivko v katero shranimo besedo katero iščemo
beseda = "lokomotiva"
print(f"Iskana beseda je: {beseda}")

st_ugibov = 10

ugib_0 = ""
ugib_1 = ""
ugib_2 = ""
ugib_3 = ""
ugib_4 = ""
ugib_5 = ""
ugib_6 = ""
ugib_7 = ""
ugib_8 = ""
ugib_9 = ""

counter = 0
while counter < st_ugibov:
    # 1. Izpišemo _ _ _ _ _ _ (in dodamo katere črke so bile uganjene)
    print()

    # 2. Uporabnika uprašamo naj vtipka katero črko ugiba
    if counter == 0:
        print(f"Nisi še ugibal.")
    elif counter == 1:
        print(f"Dosedanji ugibi: {ugib_0}")
    elif counter == 2:
        print(f"Dosedanji ugibi: {ugib_0}, {ugib_1}")
    elif counter == 3:
        print(f"Dosedanji ugibi: {ugib_0}, {ugib_1}, {ugib_2}")
    elif counter == 4:
        print(f"Dosedanji ugibi: {ugib_0}, {ugib_1}, {ugib_2}, {ugib_3}")
    elif counter == 5:
        print(f"Dosedanji ugibi: {ugib_0}, {ugib_1}, {ugib_2}, {ugib_3}, {ugib_4
    elif counter == 6:
        print(f"Dosedanji ugibi: {ugib_0}, {ugib_1}, {ugib_2}, {ugib_3}, {ugib_4
    elif counter == 7:
        print(f"Dosedanji ugibi: {ugib_0}, {ugib_1}, {ugib_2}, {ugib_3}, {ugib_4
    elif counter == 8:
        print(f"Dosedanji ugibi: {ugib_0}, {ugib_1}, {ugib_2}, {ugib_3}, {ugib_4
    elif counter == 9:
        print(f"Dosedanji ugibi: {ugib_0}, {ugib_1}, {ugib_2}, {ugib_3}, {ugib_4

    ugib = input("Vnesi ugibano črko: ")
    print(f"Ugibalo se je črko {ugib}")

    if counter == 0:
        ugib_0 = ugib
    elif counter == 1:
        ugib_1 = ugib
    elif counter == 2:
        ugib_2 = ugib
    elif counter == 3:
```

```
        ugib_3 = ugib
    elif counter == 4:
        ugib_4 = ugib
    elif counter == 5:
        ugib_5 = ugib
    elif counter == 6:
        ugib_6 = ugib
    elif counter == 7:
        ugib_7 = ugib
    elif counter == 8:
        ugib_8[Vizualizacija kode](https://pythontutor.com/render.html#mode=disp

    # 3. Preverimo ali je črka del besede
    if ugib in beseda:
        print("Pravilna črka")
    else:
        print("Nepravilna črka")

    # 4. Preverimo ali je uganil besedo:

    # 4.a   Če je uganil besedo je zmagal. Konec igre

    # 4.b   Če ni uganil besede in nima več možnih ugibov je izgubil

    # 4.c   Če ni uganil besede in ima še ugibe gremo na korak 1.

    counter += 1
```

```
Iskana beseda je: lokomotiva

Nisi še ugibal.
Vnesi ugibano črko: l
Ugibalo se je črko l
Pravilna črka

Dosedanji ugibi: l
Vnesi ugibano črko: d
Ugibalo se je črko d
Nepravilna črka

Dosedanji ugibi: l, d
Vnesi ugibano črko: e
Ugibalo se je črko e
Nepravilna črka

Dosedanji ugibi: l, d, e
Vnesi ugibano črko: e
Ugibalo se je črko e
Nepravilna črka

Dosedanji ugibi: l, d, e, e
Vnesi ugibano črko: d
Ugibalo se je črko d
Nepravilna črka

Dosedanji ugibi: l, d, e, e, d
Vnesi ugibano črko: f
Ugibalo se je črko f
Nepravilna črka

Dosedanji ugibi: l, d, e, e, d, f
Vnesi ugibano črko: g
Ugibalo se je črko g
Nepravilna črka

Dosedanji ugibi: l, d, e, e, d, f, g
Vnesi ugibano črko: b
Ugibalo se je črko b
Nepravilna črka

Dosedanji ugibi: l, d, e, e, d, f, g, b
Vnesi ugibano črko: s
Ugibalo se je črko s
Nepravilna črka

Dosedanji ugibi: l, d, e, e, d, f, g, b, s
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka
```

Vendar pa bi tak program hitro ratal nepregleden.

Problem bi se tudi pojavil kadar bi želeli povečati število ugibov katere lahko naredi igralec.

Da rešimo ta problem potrebujemo neko spremenljivko v katero lahko shranimo poljubno število vrednosti.

V pythonu takim spremenljivkam rečemo **list**. V drugih jezikih je poznana tudi kot **array**.

List

List je zbirka različnih objektov / vrednosti.

V Pythonu je list definiran z oglatimi oklepaji `[ ]`, elementi v listu pa so ločeni z vejico `,`.

In [9]:
```
živali  = ["pingvin", "medved", "los", "volk"]
print(type(živali), živali)
```

```
<class 'list'> ['pingvin', 'medved', 'los', 'volk']
```

Glavne karakteristike list-ov so:

- Lists are ordered
- Lists can contain any arbitrary objects.
- List elements can be accessed by index.
- Lists can be nested to arbitrary depth.
- Lists are mutable.
- Lists are dynamic.

Lists are ordered

To pomeni, da so podatki shranjenji v list v določenem zaporedju in ostanejo v tem zaporedju.

In [10]:
```
a  = ["pingvin", "medved", "los", "volk"]
b  = ["los", "medved", "pingvin", "volk"]
a == b # čeprov mata list a in v enake elemente, niso v istem zaporedju zato nis
```

Out[10]:
```
False
```

Lists Can Contain Arbitrary Objects

Za podatke v list-u ni potrebno, da so istega tipa (data type).

In [11]:
```
a = [21.42, "medved", 3, 4, "volk", False, 3.14159]
a
```

Out[11]:
```
[21.42, 'medved', 3, 4, 'volk', False, 3.14159]
```

Podatki v list-u se lahko podvajajo.

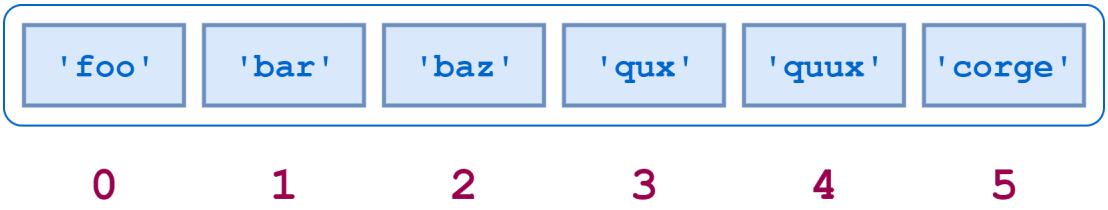
In [12]:
```
a  = ["pingvin", "medved", "los", "volk", "medved"]
a
```

Out[12]:
```
['pingvin', 'medved', 'los', 'volk', 'medved']
```

List Elements Can Be Accessed by Index

In [13]:
```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
```

Do elementov v list-u lahko dostopamo, če vemo njegov index (na kateri poziciji je).

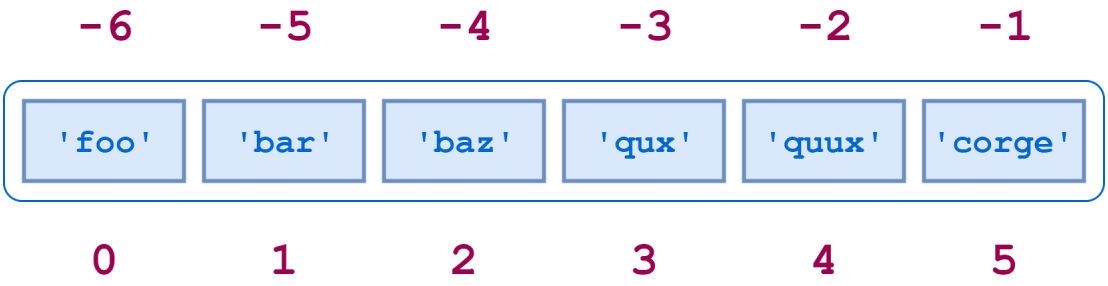


V Pythonu se indexiranje začne z 0.

In [14]:
```
print(a[0])
print(a[2])
print(a[3])
```

```
foo
baz
qux
```

Indexiramo lahko tudi z negativnimi vrednostmi:



In [15]:
```
print(a[-6])
print(a[-1])
```

```
foo
corge
```

Slicing

To nam pomaga pridobiti določene pod-liste iz že narejene list-e.

In [16]:
```
print(a[2:5])
# a[m:n] nam vrne list vrednosti, ki se nahajajo v a od vključno indexa m do izv
# a[2:5] nam vrne elemente v listu a od vključno 2 do ne vključno 5
```

```
['baz', 'qux', 'quux']
```

In [17]:
```
print(a[-5:-2]) # isto deluje z negativnimi indexi
```

```
['bar', 'baz', 'qux']
```

In [18]:
```
print(a[:4]) # če izvzamemo začetni index nam začne pri indexu 0
```



```
['foo', 'bar', 'baz', 'qux']
```

In [19]:
```
print(a[2:]) # če izvzamemo zadnji index se sprehodi do konca seznama
```

```
['baz', 'qux', 'quux', 'corge']
```

Specificeramo lahko tudi korak, za koliko naj se premakne.

In [20]:
```
print(a[::2]) # začne pri indexu 0, do konca, vsako drugo vrednost
```

```
['foo', 'baz', 'quux']
```

In [21]:
```
print(a[1:5:2])
print(a[6:0:-2]) # korak je lahko tudi negativen
print(a[::-1]) # sintaksa za sprehajanje po listu v obratnem vrstnem redu
```

```
['bar', 'qux']
['corge', 'qux', 'bar']
['corge', 'quux', 'qux', 'baz', 'bar', 'foo']
```

Use the * operator to represent the "rest" of a list

Often times, especially when dealing with the arguments to functions, it's useful to extract a few elements at the beginning (or end) of a list while keeping the "rest" for use later. Python 2 has no easy way to accomplish this aside from using slices as shown below. Python 3 allows you to use the * operator on the left hand side of an assignment to represent the rest of a sequence.

In [22]:
```
some_list = ['a', 'b', 'c', 'd', 'e']
(first, second, *rest) = some_list
print(rest)
(first, *middle, last) = some_list
print(middle)
(*head, second_last, last) = some_list
print(head)
```

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

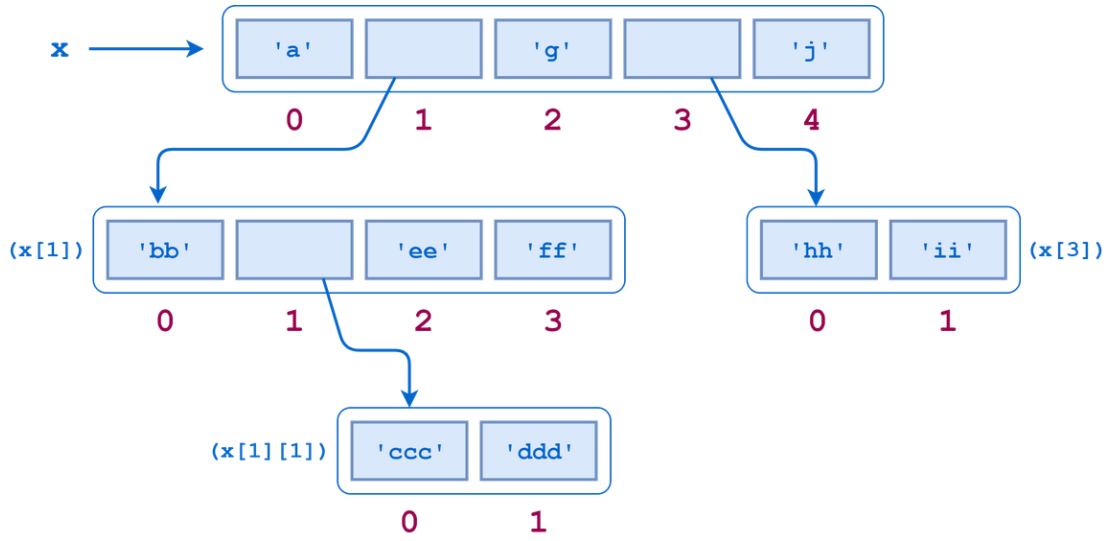
Lists can be nested to arbitrary depth

Elementi v listu so lahko poljubnega data type.

Lahko je tudi še en list. Tako lahko dodajamo dimenzije našemu list-u

In [23]:
```
x = ['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']
print(x)
```

```
['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']
```



In [24]:
```
print(x[2]) # element na indexu 2 je preprosti string dolžine 1 črke
```

```
g
```

In [25]:
```
print(x[1]) # 1 element je nov list z 4 elementi
```

```
['bb', ['ccc', 'ddd'], 'ee', 'ff']
```

In [26]:
```
print(x[1][0]) # da pridemo do njihovih elementov preprosto dodamo nov []
```

```
bb
```

In [27]:
```
print(x[1][1])
print(x[1][1][0])
```

```
['ccc', 'ddd']
ccc
```

Lists Are Mutable

To pomeni, da jih lahko spreminjamo. Lahko dodajamo elemente, jih brišemo, premikamo vrstni red, itd..

> Most of the data types you have encountered so far have been atomic types. Integer or float objects, for example, are primitive units that can't be further broken down. These types are immutable, meaning that they can't be changed once they have been assigned. It doesn't make much sense to think of changing the value of an integer. If you want a different integer, you just assign a different one.

> By contrast, the string type is a composite type. Strings are reducible to smaller parts—the component characters. It might make sense to think of changing the characters in a string. But you can't. In Python, strings are also immutable.

Spreminjanje vrednosti elementa.


```
, "medved", "los", "volk"]
print(a)
```

```
a[2] = "koza"
print(a)
```

```
['pingvin', 'medved', 'los', 'volk']
['pingvin', 'medved', 'koza', 'volk']
```

Brisanje elementa.

In [29]:
```
a  = ["pingvin", "medved", "los", "volk"]
del a[3]
print(a)
```

```
['pingvin', 'medved', 'los']
```

Spreminjanje večih elementov naenkrat.

Velikost dodanih elementov ni potrebno, da je ista kot velikost zamenjanih elementov. Python bo povečal oziroma zmanjšal list po potrebi.

In [30]:
```
a  = ["pingvin", "medved", "los", "volk"]
print(a)

a  = ["pingvin", "medved", "los", "volk"]
a[1:3] = [1.1, 2.2, 3.3, 4.4, 5.5]
print(a)

a  = ["pingvin", "medved", "los", "volk"]
a[1:4] = ['krava']
print(a)

a  = ["pingvin", "medved", "los", "volk"]
a[1:3] = [] # slicane elemente zamenjamo z praznim listom -> jih izbrišemo
print(a)
```

```
['pingvin', 'medved', 'los', 'volk']
['pingvin', 1.1, 2.2, 3.3, 4.4, 5.5, 'volk']
['pingvin', 'krava']
['pingvin', 'volk']
```

Dodajanje elementov.

Lahko dodajamo vrednosti s pomočjo .append() funkcije

In [31]:
```
a  = ["pingvin", "medved", "los", "volk"]
a.append(123)
print(a)
```

```
['pingvin', 'medved', 'los', 'volk', 123]
```

.append() doda celotno vrednost na konec lista.

In [32]:
```
a  = ["pingvin", "medved", "los", "volk"]
a.append([1, 2, 3])
print(a)
```

```
['pingvin', 'medved', 'los', 'volk', [1, 2, 3]]
```

Če želimo dodati vsako vrednost posebej lahko uporabimo .extend()

In [33]:
```
a  = ["pingvin", "medved", "los", "volk"]
a.extend([1, 2, 3])
print(a)
```

```
['pingvin', 'medved', 'los', 'volk', 1, 2, 3]
```

Dodajanje elementa na specifično mesto

`a.insert(<index>, <obj>)`
Element na mestu index zamenjamo z object.

In [34]:
```
a  = ["pingvin", "medved", "los", "volk"]
a.insert(3, 3.14159)
print(a)
```

```
['pingvin', 'medved', 'los', 3.14159, 'volk']
```

`a.remove(<obj>)`
Odstranimo object iz liste.

In [35]:
```
a  = ["pingvin", "medved", "los", "volk"]
a.remove("los")
print(a)
```

```
['pingvin', 'medved', 'volk']
```

`a.pop(index=-1)`
Odstranimo element z indexa. Metoda nam vrne izbrisani element. Default pop je zadnji element.

In [36]:
```
a  = ["pingvin", "medved", "los", "volk"]
default_pop = a.pop()
naslednji_pop = a.pop(1)

print(a)
print(default_pop)
print(naslednji_pop)
```

```
['pingvin', 'los']
volk
medved
```

Lists Are Dynamic

Dynamic pove, da ni treba na začetku definirat, da bo to list.

In [37]:
```
a  = ["pingvin", "medved", "los", "volk"]
print(a)
print(type(a))

a = 1
print(a)
print(type(a))
```

```
['pingvin', 'medved', 'los', 'volk']
<class 'list'>
1
```

In [ ]:

Vaja

Naloga:
Iz sledečega list-a pridobite vrednost **ffff**

```
our_list = ["a", ["bb", "cc"], "d", [["eee"], ["ffff"], "ggg"]]
```

In [38]:
```
our_list = ["a", ["bb", "cc"], "d", [["eee"], ["ffff"], "ggg"]]

print(our_list)
print(our_list[3])
print(our_list[3][1])
print(our_list[3][1][0])
```

```
['a', ['bb', 'cc'], 'd', [['eee'], ['ffff'], 'ggg']]
[['eee'], ['ffff'], 'ggg']
['ffff']
ffff
```

Vaja

Naloga:
Pri sledečem list-u začnite z vrednostjo 4 in vzemite vsako 3 vrednost.

```
our_list = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
```

Rešitev:
```
[4, 7, 10, 13, 16, 19]
```

In [39]:
```
our_list = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

our_sublist = (our_list[3::3])
print(our_sublist)
```

```
[4, 7, 10, 13, 16, 19]
```

Vaja

Naloga:
Uporabnik naj vnese željeno dolžino Fibonaccijevega zaporedja. Program naj nato to zaporedje shrani v list in ga na koncu izpiše.

```
Fibonacci sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34
```

In [3]:
```
x = int(input("Dolžina Fibonnacijevega zaporedja: "))
fibonacci = [0, 1]
counter = 2

while counter < x: # while len(fibonacci) < x  bi tud šlo
    fibonacci.append(fibonacci[-1] + fibonacci[-2])
    counter += 1
print(fibonacci)
```

```
Dolžina Fibonnacijevega zaporedja: 10
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Tuples

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

Definira se jih z navadnimi oklepaji `( )`.

In [60]:
```
t  = ("pingvin", "medved", "los", "volk")
print(type(t), t)
```

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

Tuples are ordered

In [61]:
```
# 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
```

Tuples can contain any arbitrary object

In [62]:
```
# Primer: Touples can contain any arbitrary object
t  = ("pingvin", "medved", "los", "volk", 1.23, True)
print(t)
```

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

Values in tuples are accessed through index

In [63]:
```
# 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')
```

Tuples can be nested

In [64]:
```
# Primer: Touples can be nested
t  = ("pingvin", "medved", "los", "volk", ("lisica", "krava"))
print(t)
```

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

Tuples are IMMUTABLE

In [140…
```
# Primer: Touples are IMMUTABLE
t = ("pingvin", "medved", "los", "volk")
t[1] = "Bork!"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-140-1f7dd710d595> 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
```

Tuples are dynamic

In [65]:
```
# 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.

In [66]:
```
t = (1,2,3,4) # nebi smel bit problem
print(t)
print(type(t))
print()

mel 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, Py
the expression (2) as simply the integer 2 and creates an int object. To tell Py
to define a singleton tuple, include a trailing comma (,) just before the closin
'''

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

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

()
<class 'tuple'>

2
<class 'int'>

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

Poleg listov in tuplov poznamo v pythonu še en datatip, imenovam **set**.

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:)

In [113…

```
s = {"medved", "zajec", "volk", "slon", "zajec"} # zajec se ne ponovi ampak je s
print(s)
print(type(s))
```

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

Elementi so lahko poljubni ampak morajo biti "immutable."

In [114…
```
s = {42, 'eee', (1, 2, 3), 3.14159} # touple je lahko element, ker je immutable
print(s)
```

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

In [115…
```
s = {11, [1, 2, 3], 'eeee'} # list ne more bit element, ker je mutable
print(x)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-115-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'
```

In [170…
```
# Primer: Can be nested
s = {42, 'eee', (1, 2, (4, 5, 6)), 3.14159}
print(s)
```

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

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.

In [171…
```
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".

In [172…
```
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-172-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.

In [173…
```
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.

In [174…
```
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.

In [175…
```
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.

In [176…
```
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/

Modifying Sets

Set-e lahko tudi spreminjamo.

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

In [177…
```
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:`

In [178…
```
x = {1, 2, 3, 4}
print(x.remove(3))
print(x)
```

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

In [179…
```
x = {1, 2, 3, 4}
x.remove(6) # gives error
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-179-4f450e54381b> 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 quietly does nothing instead of raising an exception:`

In [180…
```
x = {1, 2, 3, 4}
print(x.discard(2))
print(x)
```

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

In [181…
```
x = {1, 2, 3, 4}
x.discard(6) # doesnt give error
print(x)
```

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

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

In [182…
```
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-182-594f9b06ae7d> 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

In [183…
```
x = {1, 2, 3, 4}
print(x)
x.clear()
print(x)
```

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

Frozen sets

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

In [184…
```
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-184-769f87059d2e> in <module>
      1 x = frozenset([1, 2, 3, 4])
      2 print(x | {9, 8, 7}) # you can perform non-modifying operations on froz
ensets
----> 3 x.add(100) # should give error, because frozenset is immutable

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

Vislice

Vizualizacija kode

Naloga:
Ustvarite nov list v katerega bomo shranjevali vsako ugibano črko in te nato tudi vsakič izpišite.

```
Iskana beseda je: lokomotiva

1 / 10 ugib
Dosedanji ugibi []
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

2 / 10 ugib
Dosedanji ugibi ['a']
Vnesi ugibano črko: b
Ugibalo se je črko b
Nepravilna črka

3 / 10 ugib
Dosedanji ugibi ['a', 'b']
Vnesi ugibano črko: c
Ugibalo se je črko c
Nepravilna črka

4 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c']
Vnesi ugibano črko: d
Ugibalo se je črko d
Nepravilna črka

5 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'd']
Vnesi ugibano črko: f
Ugibalo se je črko f
Nepravilna črka
```

```
Dosedanji ugibi ['a', 'b', 'c', 'd', 'f']
Vnesi ugibano črko: g
Ugibalo se je črko g
Nepravilna črka

7 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'd', 'f', 'g']
Vnesi ugibano črko: h
Ugibalo se je črko h
Nepravilna črka

8 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'd', 'f', 'g', 'h']
Vnesi ugibano črko: z
Ugibalo se je črko z
Nepravilna črka

9 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'd', 'f', 'g', 'h', 'z']
Vnesi ugibano črko: t
Ugibalo se je črko t
Pravilna črka

10 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'd', 'f', 'g', 'h', 'z', 't']
Vnesi ugibano črko: r
Ugibalo se je črko r
Nepravilna črka
```

In [59]:

```
# Ustvarimo spremenljivko v katero shranimo besedo katero iščemo
beseda = "lokomotiva"
print(f"Iskana beseda je: {beseda}")

st_ugibov = 10

ugibi = []

counter = 0
while counter < st_ugibov:
    # 1. Izpišemo _ _ _ _ _ _ (in dodamo katere črke so bile uganjene)
    print()
    print(f"{counter+1} / {st_ugibov} ugib")
    print(f"Dosedanji ugibi {ugibi}")

    ugib = input("Vnesi ugibano črko: ")
    print(f"Ugibalo se je črko {ugib}")
    ugibi.append(ugib)

    # 3. Preverimo ali je črka del besede
    if ugib in beseda:
        print("Pravilna črka")
    else:
        print("Nepravilna črka")

    # 4. Preverimo ali je uganil besedo:

    # je uganil besedo je zmagal. Konec igre
```

```
    # 4.b   Če ni uganil besede in nima več možnih ugibov je izgubil

    # 4.c   Če ni uganil besede in ima še ugibe gremo na korak 1.

    counter += 1
```

```
Iskana beseda je: lokomotiva

1 / 10 ugib
Dosedanji ugibi []
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

2 / 10 ugib
Dosedanji ugibi ['a']
Vnesi ugibano črko: b
Ugibalo se je črko b
Nepravilna črka

3 / 10 ugib
Dosedanji ugibi ['a', 'b']
Vnesi ugibano črko: c
Ugibalo se je črko c
Nepravilna črka

4 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c']
Vnesi ugibano črko: d
Ugibalo se je črko d
Nepravilna črka

5 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'd']
Vnesi ugibano črko: f
Ugibalo se je črko f
Nepravilna črka

6 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'd', 'f']
Vnesi ugibano črko: g
Ugibalo se je črko g
Nepravilna črka

7 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'd', 'f', 'g']
Vnesi ugibano črko: h
Ugibalo se je črko h
Nepravilna črka

8 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'd', 'f', 'g', 'h']
Vnesi ugibano črko: z
Ugibalo se je črko z
Nepravilna črka

9 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'd', 'f', 'g', 'h', 'z']
Vnesi ugibano črko: t
Ugibalo se je črko t
Pravilna črka

10 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'd', 'f', 'g', 'h', 'z', 't']
Vnesi ugibano črko: r
```

```
Ugibalo se je črko r
Nepravilna črka
```

While zanka se uporablja za ponavljanje kode, dokler se ne izpolni specifični pogoj.

Poleg nje imamo v pythonu še eno zanko imenovano **for loop**, ki pa neko kodo ponovi specifično število korakov.

For loop

Uporablja se kadar hočemo izvesti blok kode za vnaprej določeno število ponovitev.

Primer: kadar hočemo izvesti blok kode za vsak element v list-u.

```
for <var> in <iterable>:
    <statement(s)>
```

In [17]:
```
primes = [2, 3, 5, 7, 11] #itrable
for prime in primes:
    print(f'{prime} is  a prime number.')
```

```
2 is  a prime number.
3 is  a prime number.
5 is  a prime number.
7 is  a prime number.
11 is  a prime number.
```

Vizualizacija kode

In [42]:
```
kid_ages = (3, 7, 12)
for age in kid_ages:
    print(f'I have a {age} year old kid.')
```

```
I have a 3 year old kid.
I have a 7 year old kid.
I have a 12 year old kid.
```

Vizualizacija kode

Velikokrat se skupaj z for-loop uporablja funkcija range().

`range(start, stop, step)`

- start - Optional. An integer number specifying at which position to start. Default is 0
- stop - An integer number specifying at which position to end, excluding this number.
- step - Optional. An integer number specifying the incrementation. Default is 1

Funkcija range nam zgenerira list števil.

In [24]:
```
x = range(5, 10, 1)
print(type(x))
```

```
print(list(x))
```

```
<class 'range'>
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [43]:
```
n_steps = 5
for i in range(n_steps):
    print(f"Smo na koraku: {i}")
```

```
Smo na koraku: 0
Smo na koraku: 1
Smo na koraku: 2
Smo na koraku: 3
Smo na koraku: 4
```

Vizualizacija kode

Vaja

Naloga:

Podane imate podatke.

```
country_gdp = [['United States', 22996100, 22145244],
               ['China', 17734063, 17077902],
               ['Japan', 4937422, 4754737],
               ['Germany', 4223116, 4066860],
               ['United Kingdom', 3186860, 3068946],
               ['India', 3173398, 3055982],
               ['France', 2937473, 2828786],
               ['Italy', 2099880, 2022184],
               ['Canada', 1990762, 1917103],
               ['Korea, Rep.', 1798534, 1731988]]
```

Prva vrednost je ime države, druga vrednost je GDP države v $, tretja vrednost je GDP države v €.

Za vsako državo izpišite njeno ime in koliko je njen gdp v €.

```
GDP države United States je 22145244 €
GDP države China je 17077902 €
GDP države Japan je 4754737 €
GDP države Germany je 4066860 €
GDP države United Kingdom je 3068946 €
GDP države India je 3055982 €
GDP države France je 2828786 €
GDP države Italy je 2022184 €
GDP države Canada je 1917103 €
GDP države Korea, Rep. je 1731988 €
```

Vizualizacija kode


```
['United States', 22996100, 22145244],
['China', 17734063, 17077902],
```

```
                    ['Japan', 4937422, 4754737],
                    ['Germany', 4223116, 4066860],
                    ['United Kingdom', 3186860, 3068946],
                    ['India', 3173398, 3055982],
                    ['France', 2937473, 2828786],
                    ['Italy', 2099880, 2022184],
                    ['Canada', 1990762, 1917103],
                    ['Korea, Rep.', 1798534, 1731988]]

for country in country_gdp:
    print(f"GDP države {country[0]} je {country[-1]} €")
```

```
GDP države United States je 22145244 €
GDP države China je 17077902 €
GDP države Japan je 4754737 €
GDP države Germany je 4066860 €
GDP države United Kingdom je 3068946 €
GDP države India je 3055982 €
GDP države France je 2828786 €
GDP države Italy je 2022184 €
GDP države Canada je 1917103 €
GDP države Korea, Rep. je 1731988 €
```

> **Nasveti**

- Use the enumerate function in loops instead of creating an “index” variable

Programmers coming from other languages are used to explicitly declaring a variable to track the index of a container in a loop. For example, in C++:

```
for (int i=0; i < container.size(); ++i)
{
// Do stuff
}
```

In Python, the enumerate built-in function handles this role.

In [7]:
```
moj_list = ["Anže", "Luka", "Mojca"]
index = 0
for element in moj_list:
    print (f'{index} {element}')
    index += 1
```

```
0 Anže
1 Luka
2 Mojca
```

In [6]:
```
#Idiomatic
moj_list = ["Anže", "Luka", "Mojca"]
for index, element in enumerate(moj_list):
    print (f'{index} {element}')
```

```
0 Anže
1 Luka
2 Mojca
```

Vislice

Naloga:
Spremenimo našo kodo, da bo uporabljala for loop, sam imamo točno podano kolikokrat želimo ponoviti kodo.

```
Iskana beseda je: lokomotiva

1 / 10 ugib
Dosedanji ugibi []
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

2 / 10 ugib
Dosedanji ugibi ['a']
Vnesi ugibano črko: b
Ugibalo se je črko b
Nepravilna črka

3 / 10 ugib
Dosedanji ugibi ['a', 'b']
Vnesi ugibano črko: c
Ugibalo se je črko c
Nepravilna črka

4 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c']
Vnesi ugibano črko: v
Ugibalo se je črko v
Pravilna črka

5 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'v']
Vnesi ugibano črko: d
Ugibalo se je črko d
Nepravilna črka

6 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'v', 'd']
Vnesi ugibano črko: e
Ugibalo se je črko e
Nepravilna črka

7 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'v', 'd', 'e']
Vnesi ugibano črko: f
Ugibalo se je črko f
Nepravilna črka

8 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'v', 'd', 'e', 'f']
```

```
Vnesi ugibano črko: h
Ugibalo se je črko h
Nepravilna črka

9 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'v', 'd', 'e', 'f', 'h']
Vnesi ugibano črko: i
Ugibalo se je črko i
Pravilna črka

10 / 10 ugib
Dosedanji ugibi ['a', 'b', 'c', 'v', 'd', 'e', 'f', 'h', 'i']
Vnesi ugibano črko: j
Ugibalo se je črko j
Nepravilna črka
```

Vizualizacija kode

In [ ]:
```python
# Ustvarimo spremenljivko v katero shranimo besedo katero iščemo
beseda = "lokomotiva"
print(f"Iskana beseda je: {beseda}")

st_ugibov = 10

ugibi = []

for i in range(st_ugibov):
    print()

    # 1. Izpišemo _ _ _ _ _ _ (in dodamo katere črke so bile uganjene)
    print(f"{i+1} / {st_ugibov} ugib")
    print(f"Dosedanji ugibi {ugibi}")

    ugib = input("Vnesi ugibano črko: ")
    print(f"Ugibalo se je črko {ugib}")
    ugibi.append(ugib)

    # 3. Preverimo ali je črka del besede
    if ugib in beseda:
        print("Pravilna črka")
    else:
        print("Nepravilna črka")

    # 4. Preverimo ali je uganil besedo:

    # 4.a   Če je uganil besedo je zmagal. Konec igre

    # 4.b   Če ni uganil besede in nima več možnih ugibov je izgubil

    # 4.c   Če ni uganil besede in ima še ugibe gremo na korak 1.
```

Dodajmo sedaj še izpis iskane besede:

Naloga:
Dodajte kodo, da se vsak korak zanke izpiše tudi iskana beseda. Črke katere so bile pravilno uganjene naj se izpišejo, črke katere še nismo uganili naj se prikažejo kot `_`.

```
Iskana beseda je: lokomotiva


_ _ _ _ _ _ _ _ _ _
1 / 10 ugib
Dosedanji ugibi []
Vnesi ugibano črko: l
Ugibalo se je črko l
Pravilna črka

l_ _ _ _ _ _ _ _ _
2 / 10 ugib
Dosedanji ugibi ['l']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

l_ _ _ _ _ _ _ _ a
3 / 10 ugib
Dosedanji ugibi ['l', 'a']
Vnesi ugibano črko: s
Ugibalo se je črko s
Nepravilna črka

l_ _ _ _ _ _ _ _ a
4 / 10 ugib
Dosedanji ugibi ['l', 'a', 's']
Vnesi ugibano črko: d
Ugibalo se je črko d
Nepravilna črka

l_ _ _ _ _ _ _ _ a
5 / 10 ugib
Dosedanji ugibi ['l', 'a', 's', 'd']
Vnesi ugibano črko: o
Ugibalo se je črko o
Pravilna črka

lo_ o_ o_ _ _ a
6 / 10 ugib
Dosedanji ugibi ['l', 'a', 's', 'd', 'o']
Vnesi ugibano črko: k
Ugibalo se je črko k
Pravilna črka

loko_ o_ _ _ a
7 / 10 ugib
Dosedanji ugibi ['l', 'a', 's', 'd', 'o', 'k']
Vnesi ugibano črko: g
Ugibalo se je črko g
```

```
Nepravilna črka

loko_ o_ _ _ a
8 / 10 ugib
Dosedanji ugibi ['l', 'a', 's', 'd', 'o', 'k', 'g']
Vnesi ugibano črko: b
Ugibalo se je črko b
Nepravilna črka

loko_ o_ _ _ a
9 / 10 ugib
Dosedanji ugibi ['l', 'a', 's', 'd', 'o', 'k', 'g', 'b']
Vnesi ugibano črko: v
Ugibalo se je črko v
Pravilna črka

loko_ o_ _ va
10 / 10 ugib
Dosedanji ugibi ['l', 'a', 's', 'd', 'o', 'k', 'g', 'b', 'v']
Vnesi ugibano črko: c
Ugibalo se je črko c
Nepravilna črka
```

Vizualizacija kode

In [68]:

```
# Ustvarimo spremenljivko v katero shranimo besedo katero iščemo
beseda = "lokomotiva"
print(f"Iskana beseda je: {beseda}")

st_ugibov = 10

ugibi = []

for i in range(st_ugibov):
    print()
    # 1. Izpišemo _ _ _ _ _ _ (in dodamo katere črke so bile uganjene)
    izpis = ""
    for ch in beseda:
        if ch in ugibi:
            izpis += ch
        else:
            izpis += "_ "
    print(izpis)
    print(f"{i+1} / {st_ugibov} ugib")
    print(f"Dosedanji ugibi {ugibi}")

    ugib = input("Vnesi ugibano črko: ")
    print(f"Ugibalo se je črko {ugib}")
    ugibi.append(ugib)

    # 3. Preverimo ali je črka del besede
    if ugib in beseda:
        print("Pravilna črka")
    else:
        print("Nepravilna črka")

# 4. Preverimo ali je uganil besedo:
```

```
    # 4.a   Če je uganil besedo je zmagal. Konec igre

    # 4.b   Če ni uganil besede in nima več možnih ugibov je izgubil

    # 4.c   Če ni uganil besede in ima še ugibe gremo na korak 1.
```

```
Iskana beseda je: lokomotiva


_ _ _ _ _ _ _ _ _ _
1 / 10 ugib
Dosedanji ugibi []
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

_ _ _ _ _ _ _ _ _ a
2 / 10 ugib
Dosedanji ugibi ['a']
Vnesi ugibano črko: l
Ugibalo se je črko l
Pravilna črka

l_ _ _ _ _ _ _ _ a
3 / 10 ugib
Dosedanji ugibi ['a', 'l']
Vnesi ugibano črko: k
Ugibalo se je črko k
Pravilna črka

l_ k_ _ _ _ _ _ a
4 / 10 ugib
Dosedanji ugibi ['a', 'l', 'k']
Vnesi ugibano črko: c
Ugibalo se je črko c
Nepravilna črka

l_ k_ _ _ _ _ _ a
5 / 10 ugib
Dosedanji ugibi ['a', 'l', 'k', 'c']
Vnesi ugibano črko: o
Ugibalo se je črko o
Pravilna črka

loko_ o_ _ _ a
6 / 10 ugib
Dosedanji ugibi ['a', 'l', 'k', 'c', 'o']
Vnesi ugibano črko: m
Ugibalo se je črko m
Pravilna črka

lokomo_ _ _ a
7 / 10 ugib
Dosedanji ugibi ['a', 'l', 'k', 'c', 'o', 'm']
Vnesi ugibano črko: n
Ugibalo se je črko n
Nepravilna črka

lokomo_ _ _ a
8 / 10 ugib
Dosedanji ugibi ['a', 'l', 'k', 'c', 'o', 'm', 'n']
Vnesi ugibano črko: b
Ugibalo se je črko b
Nepravilna črka
```



```
9 / 10 ugib
```

```
Dosedanji ugibi ['a', 'l', 'k', 'c', 'o', 'm', 'n', 'b']
Vnesi ugibano črko: v
Ugibalo se je črko v
Pravilna črka

lokomo_ _ va
10 / 10 ugib
Dosedanji ugibi ['a', 'l', 'k', 'c', 'o', 'm', 'n', 'b', 'v']
Vnesi ugibano črko: d
Ugibalo se je črko d
Nepravilna črka
```

Za konec dodajmo še kodo, ki preveri ali je igralec zmagal ali ne.

Za vsako črko bomo preverili ali smo jo že uganili ali ne. Če smo jo uganili bomo nekam shranili vrednost `True`, če je še nismo bomo shranili `False`.

To bomo shranili v data tip **dictionary**.

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.

In [1]:
```
d = {
'macek' : 'Silvestre',
'pes'   : 'Fido',
'papagaj': 'Kakadu'
}
print(d)
print(type(d))
```

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

In [2]:
```
# Primer: Can contain any arbitrary objects
d = {
'macek' : 1,
'pes'   : 'Fido',
'papagaj': False
}
print(d)
```

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

Accessing dictionary value

Vrednosti najdemo preko ključev.

In [3]:
```
d = {
'macek' : 'Silvestre',
'pes'   : 'Fido',
'papagaj': 'Kakadu'
}
print(d['papagaj'])
```

```
Kakadu
```

Če vpišemo ključ, ki ne obstaja python vrne napako.

In [4]:
```
d = {
'macek' : 'Silvestre',
'pes'   : 'Fido',
'papagaj': 'Kakadu'
}
d['koza'] # should give KeyError
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In [4], line 6
      1 d = {
      2 'macek' : 'Silvestre',
      3 'pes'   : 'Fido',
      4 'papagaj': 'Kakadu'
      5 }
----> 6 d['koza'] # should give KeyError

KeyError: 'koza'
```

Dodajanje novih vrednosti

In [5]:
```
d = {
'macek' : 'Silvestre',
'pes'   : 'Fido',
'papagaj': 'Kakadu'
}
d['koza'] = "Micka"
print(d)
```

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

Posodabljanje vrednosti.

In [6]:
```
d['koza'] = 'Helga'
print(d)
```

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

Brisanje elementa.

In [7]:
```
del d['koza']
print(d)
```

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

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,...).

In [8]:
```
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 s
# Sej keywords dajat sm je nesmiselno
```

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

In [9]:
```
# 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)
Cell In [9], line 2
      1 # Primer: Vrže error, ker hočemo kot ključ uporabiti list, ki pa je mut
able
----> 2 d = {[1,1]: 'a', [1,2]: 'b'}
      3 d = {(1,2,[1,2]): "f",}

TypeError: unhashable type: 'list'
```

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):

In [10]:
```
d = {
'macek' : 'Silvestre',
'pes'   : 'Fido',
'papagaj': 'Kakadu',
'macek' : 'Amadeus'
}
print(d)
```

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

Built-in Dictionary Methods

Še nekaj ostalih metod.

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

In [11]:
```
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

In [12]:
```
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.

In [13]:
```
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.

In [14]:
```
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.

In [15]:
```
d = {'a': 10, 'b': 20, 'c': 30}
print(list(d.keys()))
```

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

`d.values()`
Vrne nam list vrednosti.

In [16]:
```
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.

In [17]:
```
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.

In [18]:
```
d = {'a': 10, 'b': 20, 'c': 30}
print(d.pop('z'))
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In [18], line 2
      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.

In [19]:
```
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

In [20]:
```
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.

In [21]:
```
d = {}
d.popitem()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In [21], line 2
      1 d = {}
----> 2 d.popitem()

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

In [22]:
```
# 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'>
```

Vaja

Naloga: Sledečemu dictionary zamenjajte 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
}
```

In [17]:
```
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}
```

Vaja

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"
        }
    }
}
```

In [23]:
```
d = {
    "a": "a",
    "b": "b",
```

```
        2: 22,
        3: 33,
        4: {
            5: "ccc",
            6: "ddd",
            "7": "fff"
        }
    }
}

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

```
fff
```

Poglejmo si še kako lahko iteriramo preko dictionaria

In [24]:
```
pets = {
    'macka': 6,
    'pes': 12,
    'krava': 20
}

for x in pets:
    print(type(x), x)
```

```
<class 'str'> macka
<class 'str'> pes
<class 'str'> krava
```

In [27]:
```
pets = {
    'macka': 6,
    'pes': 12,
    'krava': 20
}

keys = pets.keys()
print(type(keys), keys)
print(list(keys))

for key in pets.keys():
    print("Ključ: ", key, ", Vrednost: ", pets[key])
```

```
<class 'dict_keys'> dict_keys(['macka', 'pes', 'krava'])
['macka', 'pes', 'krava']
Ključ:  macka , Vrednost:  6
Ključ:  pes , Vrednost:  12
Ključ:  krava , Vrednost:  20
```

In [28]:
```
pets = {
    'macka': 6,
    'pes': 12,
    'krava': 20
}

values = pets.values()
print(type(values), values)
print(list(values))
```

```
for value in pets.values():
    print("Vrednost: ", value)
```

```
<class 'dict_values'> dict_values([6, 12, 20])
[6, 12, 20]
Vrednost:  6
Vrednost:  12
Vrednost:  20
```

In [29]:
```
pets = {
    'macka': 6,
    'pes': 12,
    'krava': 20
}

items = pets.items()
print(type(items), items)
print(list(items))

for item in pets.items():
    print(f"{item} -> {item[0]} {item[1]}")
```

```
<class 'dict_items'> dict_items([('macka', 6), ('pes', 12), ('krava', 20)])
[('macka', 6), ('pes', 12), ('krava', 20)]
('macka', 6) -> macka 6
('pes', 12) -> pes 12
('krava', 20) -> krava 20
```

In [31]:
```
pets = {
    'macka': 6,
    'pes': 12,
    'krava': 20
}

for key, value in pets.items():
    print("Ključ: ", key, ", Vrednost: ", value)
```

```
Ključ:  macka , Vrednost:  6
Ključ:  pes , Vrednost:  12
Ključ:  krava , Vrednost:  20
```

Vaja

Naloga:

Iz danega dictionary izpišite vse ključe, katerih vrednost vsebuje črko **r** ali **R**.

```
d = {
    "mačka": "Micka",
    "pes": "Fido",
    "volk": "Rex",
    "medved": "Žan",
    "slon": "Jan",
    "žirafa": "Helga",
    "lev": "Gašper",
    "tiger": "Anže",
    "papagaj": "Črt",
    "ribica": "Elena",
    "krokodil": "Kasper",
```

```
    "zajec": "Lars",
    "kamela": "Manca"
}
```

```
volk
lev
papagaj
krokodil
zajec
```

Vizualizacija kode

In [54]:
```
d = {
    "mačka": "Micka",
    "pes": "Fido",
    "volk": "Rex",
    "medved": "Žan",
    "slon": "Jan",
    "žirafa": "Helga",
    "lev": "Gašper",
    "tiger": "Anže",
    "papagaj": "Črt",
    "ribica": "Elena",
    "krokodil": "Kasper",
    "zajec": "Lars",
    "kamela": "Manca"
}


for key,value in d.items():
    if "r" in value or "R" in value:
        print(f"{key}")
```

```
volk
lev
papagaj
krokodil
zajec
```

Vislice

Naloga:
V naš program dodajmo kodo, ki preveri ali je igralec zmagal ali izgubil. Program naj v spremenljivko `won` shrani vrednost `True`, če je igralec zmagal, in `False`, če je igralec izgubil.

Za preverjanje lahko ustvarite novo spremenljivko `word_check`, ki je tipa `dict`. Ključi v spremenljivki so posamezne črke iskane besede. Vrednost posameznega ključa pa je `True` ali `False`, glede na to ali smo črko že našli.

Primer spremenljivke: loko_ o_ _ _ _

```
print(word_check)
{
 "l": True,
 "o": True,
 "k": True,
 "m": False,
 "t": False,
 "i": False,
 "v": False,
 "a": False
}
```

```
Iskana beseda je: lokomotiva


_ _ _ _ _ _ _ _ _ _
1 / 10 ugib
Dosedanji ugibi []
Vnesi ugibano črko: l
Ugibalo se je črko l
Pravilna črka

l_ _ _ _ _ _ _ _ _
2 / 10 ugib
Dosedanji ugibi ['l']
Vnesi ugibano črko: č
Ugibalo se je črko č
Nepravilna črka

l_ _ _ _ _ _ _ _ _
3 / 10 ugib
Dosedanji ugibi ['l', 'č']
Vnesi ugibano črko: o
Ugibalo se je črko o
Pravilna črka

lo_ o_ o_ _ _ _
4 / 10 ugib
Dosedanji ugibi ['l', 'č', 'o']
Vnesi ugibano črko: k
Ugibalo se je črko k
Pravilna črka

loko_ o_ _ _ _
5 / 10 ugib
Dosedanji ugibi ['l', 'č', 'o', 'k']
Vnesi ugibano črko: m
Ugibalo se je črko m
Pravilna črka

lokomo_ _ _ _
6 / 10 ugib
Dosedanji ugibi ['l', 'č', 'o', 'k', 'm']
Vnesi ugibano črko: t
```



```
črko t
```

```
Pravilna črka

lokomot_ _ _
7 / 10 ugib
Dosedanji ugibi ['l', 'č', 'o', 'k', 'm', 't']
Vnesi ugibano črko: v
Ugibalo se je črko v
Pravilna črka

lokomot_ v_
8 / 10 ugib
Dosedanji ugibi ['l', 'č', 'o', 'k', 'm', 't', 'v']
Vnesi ugibano črko: i
Ugibalo se je črko i
Pravilna črka

lokomotiv_
9 / 10 ugib
Dosedanji ugibi ['l', 'č', 'o', 'k', 'm', 't', 'v', 'i']
Vnesi ugibano črko: š
Ugibalo se je črko š
Nepravilna črka

lokomotiv_
10 / 10 ugib
Dosedanji ugibi ['l', 'č', 'o', 'k', 'm', 't', 'v', 'i', 'š']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka
ZMAGA
```

Vizualizacija kode

In [34]:
```
# Ustvarimo spremenljivko v katero shranimo besedo katero iščemo
beseda = "lokomotiva"
print(f"Iskana beseda je: {beseda}")

st_ugibov = 10

ugibi = []

won = False

word_check = {}
for ch in beseda:
    word_check[ch] = False

for i in range(st_ugibov):
    print()
    # 1. Izpišemo _ _ _ _ _ _ (in dodamo katere črke so bile uganjene)
    izpis = ""
    for ch in beseda:
        if ch in ugibi:
            izpis += ch
        else:
            izpis += "_ "
    print(izpis)
```

```
    print(f"{i+1} / {st_ugibov} ugib")
    print(f"Dosedanji ugibi {ugibi}")

    ugib = input("Vnesi ugibano črko: ")
    print(f"Ugibalo se je črko {ugib}")
    ugibi.append(ugib)

    # 3. Preverimo ali je črka del besede
    if ugib in beseda:
        print("Pravilna črka")
        word_check[ugib] = True
    else:
        print("Nepravilna črka")

    # 4. Preverimo ali je uganil besedo:
    # 4.a   Če je uganil besedo je zmagal. Konec igre
    # 4.b   Če ni uganil besede in nima več možnih ugibov je izgubil
    # 4.c   Če ni uganil besede in ima še ugibe gremo na korak 1.
    if False in word_check.values():
        won = False
    else:
        won = True



if won:
    print("ZMAGA")
else:
    print("Poraz.")
```

```
Iskana beseda je: lokomotiva

_ _ _ _ _ _ _ _ _ _
1 / 10 ugib
Dosedanji ugibi []
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

_ _ _ _ _ _ _ _ _ a
2 / 10 ugib
Dosedanji ugibi ['a']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

_ _ _ _ _ _ _ _ _ a
3 / 10 ugib
Dosedanji ugibi ['a', 'a']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

_ _ _ _ _ _ _ _ _ a
4 / 10 ugib
Dosedanji ugibi ['a', 'a', 'a']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

_ _ _ _ _ _ _ _ _ a
5 / 10 ugib
Dosedanji ugibi ['a', 'a', 'a', 'a']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

_ _ _ _ _ _ _ _ _ a
6 / 10 ugib
Dosedanji ugibi ['a', 'a', 'a', 'a', 'a']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

_ _ _ _ _ _ _ _ _ a
7 / 10 ugib
Dosedanji ugibi ['a', 'a', 'a', 'a', 'a', 'a']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

_ _ _ _ _ _ _ _ _ a
8 / 10 ugib
Dosedanji ugibi ['a', 'a', 'a', 'a', 'a', 'a', 'a']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

_ _ _ _ _ _ _ _ _ a
9 / 10 ugib
```

```
Dosedanji ugibi ['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

_ _ _ _ _ _ _ _ _ a
10 / 10 ugib
Dosedanji ugibi ['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka
Poraz.
```

Naša koda deluje. Problem se pojavi, če mi besedo uganemo ampak imamo na voljo še nekaj ugibov.

V našem primeru koda nadaljuje, dokler ne naredimo 10-ih ugibov. Kar bi mi želeli je, da se izvajanje zanke zaključi.

Takšno kontrolo nad koraki v zanki lahko dosežemo z **break** in **continue**.

Break

Break keyword terminira najbolj notranjo zanko v kateri se nahaja.

Vizualizacija kode

In [95]:
```
for i in range(10):
    print("i =", i)
    if i >= 5:
        break
    print("i**2=", i**2)
print("Nadaljevanje programa")
```

```
i = 0
i**2= 0
i = 1
i**2= 1
i = 2
i**2= 4
i = 3
i**2= 9
i = 4
i**2= 16
i = 5
Nadaljevanje programa
```

Vizualizacija kode

In [105…
```
# Naloga: Izpisati fibonacijevo zaporedje dokler ne pridemo do številke, ki je v
x = [0, 1]
while True:
    next_number = x[-1] + x[-2]
    x.append(next_number)
    if next_number >= 200:
```

```
        break
print(x)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233]
```

Continue

Continue keyword izpusti kodo, ki se more še izvesti, in skoči na naslednjo iteracijo zanke

.

Vizualizacija kode

In [45]:
```
for i in range(10):
    print("i = ", i)
    if i >= 5:
        continue
    print("i**2 = ", i**2)
print("Nadaljevaje programa")
```

```
Avto je ok.
Naslednji korak zanke
Avto je ok.
Naslednji korak zanke
Avto je ok.
Naslednji korak zanke
Avto je zanič.
Avto je ok.
Naslednji korak zanke
End
```

Vaja

Naloga:
Imate podane podatke iz olimpijskih iger 2021. Prva vrednost je ime države Druga vrednost je število prejetih ZLATIH medalj Tretja vrednost je število prejetih SREBRNIH medalj Četrta vrednost je število prejetih BRONASTIH medalj Peta vrednost je uvrščenost države glede na število prejetih medalj

Program naj se začne sprehajati čez podatke in naj izpiše ime države in njen rank.

Ko pride do SLOVENIJE naj izpiše njeno ime, rank, število zlatih, srebrnih in bronastih medalj.

Nato naj se izvajanje zanke zaključi.

```
United States of America, rank: 1
People's Republic of China, rank: 2
Japan, rank: 5
Great Britain, rank: 4
```

```
Australia, rank: 6
Netherlands, rank: 9
France, rank: 10
Germany, rank: 8
Italy, rank: 7
Canada, rank: 11
Brazil, rank: 12
New Zealand, rank: 13
Cuba, rank: 18
Hungary, rank: 13
Slovenia, rank: 42.      Z: 3, S: 1, B: 1
```

Vizualizacija kode

In [107…

```
data = [['United States of America', 39, 41, 33, 1],
        ["People's Republic of China", 38, 32, 18, 2],
        ['Japan', 27, 14, 17, 5],
        ['Great Britain', 22, 21, 22, 4],
        ['ROC', 20, 28, 23, 3],
        ['Australia', 17, 7, 22, 6],
        ['Netherlands', 10, 12, 14, 9],
        ['France', 10, 12, 11, 10],
        ['Germany', 10, 11, 16, 8],
        ['Italy', 10, 10, 20, 7],
        ['Canada', 7, 6, 11, 11],
        ['Brazil', 7, 6, 8, 12],
        ['New Zealand', 7, 6, 7, 13],
        ['Cuba', 7, 3, 5, 18],
        ['Hungary', 6, 7, 7, 13],
        ['Slovenia', 3, 1, 1, 42],
        ['Republic of Korea', 6, 4, 10, 13],
        ['Poland', 4, 5, 5, 19],
        ['Czech Republic', 4, 4, 3, 23],
        ['Kenya', 4, 4, 2, 25],
        ['Norway', 4, 2, 2, 29],
        ['Jamaica', 4, 1, 4, 26],
        ['Spain', 3, 8, 6, 17],
        ['Sweden', 3, 6, 0, 26],
        ['Switzerland', 3, 4, 6, 20],
        ['Denmark', 3, 4, 4, 23],
        ['Croatia', 3, 3, 2, 29],
        ['Islamic Republic of Iran', 3, 2, 2, 33],
        ['Serbia', 3, 1, 5, 26],
        ['Belgium', 3, 1, 3, 33],
        ['Bulgaria', 3, 1, 2, 39],
        ['Uzbekistan', 3, 0, 2, 42],
        ['Georgia', 2, 5, 1, 29],
        ['Chinese Taipei', 2, 4, 6, 22],
        ['Turkey', 2, 2, 9, 20],
        ['Greece', 2, 1, 1, 47],
        ['Uganda', 2, 1, 1, 47],
        ['Ecuador', 2, 1, 0, 60],
        ['Ireland', 2, 0, 2, 47],
        ['Israel', 2, 0, 2, 47],
        ['Qatar', 2, 0, 1, 60],
        ['Bahamas', 2, 0, 0, 66],
        ['Kosovo', 2, 0, 0, 66],
        ['                e', 1, 6, 12, 16],
```

```
        ['Belarus', 1, 3, 3, 33],
        ['Romania', 1, 3, 0, 47],
        ['Venezuela', 1, 3, 0, 47],
        ['India', 1, 2, 4, 33],
        ['Hong Kong, China', 1, 2, 3, 39],
        ['Philippines', 1, 2, 1, 47]]

for country in data:
    if country[0] == "Slovenia":
        print(f"{country[0]}, rank: {country[-1]}.\t Z: {country[1]}, S: {countr
        break
    print(f"{country[0]}, rank: {country[-1]}")
```

```
United States of America, rank: 1
People's Republic of China, rank: 2
Japan, rank: 5
Great Britain, rank: 4
ROC, rank: 3
Australia, rank: 6
Netherlands, rank: 9
France, rank: 10
Germany, rank: 8
Italy, rank: 7
Canada, rank: 11
Brazil, rank: 12
New Zealand, rank: 13
Cuba, rank: 18
Hungary, rank: 13
Slovenia, rank: 42.      Z: 3, S: 1, B: 1
```

Vaja

Naloga:
Poiščite vsa praštevila med 2 in 30.

```
2 JE praštevilo!
3 JE praštevilo!
5 JE praštevilo!
7 JE praštevilo!
11 JE praštevilo!
13 JE praštevilo!
17 JE praštevilo!
19 JE praštevilo!
23 JE praštevilo!
29 JE praštevilo!
```

Vizualizacija kode

In [44]:
```
for num in range(2,31):
    prime = True
    for i in range(2,num):
        #print(f"{num} / {i}. Ostanek je {num%i}")
        if (num%i==0):
            prime = False
            break
    if prime:
```

```
        print(f"{num} JE praštevilo!")
    #else:
        #print(f"{num} NI praštevilo.")
```

```
2 JE praštevilo!
3 JE praštevilo!
5 JE praštevilo!
7 JE praštevilo!
11 JE praštevilo!
13 JE praštevilo!
17 JE praštevilo!
19 JE praštevilo!
23 JE praštevilo!
29 JE praštevilo!
```

Vislice

Naloga:
Dopolnimo naš program tako, da če je igralec zmagal, program zaključi zanko.

```
Iskana beseda je: lokomotiva

_ _ _ _ _ _ _ _ _ _
1 / 10 ugib
Dosedanji ugibi []
Vnesi ugibano črko: l
Ugibalo se je črko l
Pravilna črka

l_ _ _ _ _ _ _ _ _
2 / 10 ugib
Dosedanji ugibi ['l']
Vnesi ugibano črko: o
Ugibalo se je črko o
Pravilna črka

lo_ o_ o_ _ _ _
3 / 10 ugib
Dosedanji ugibi ['l', 'o']
Vnesi ugibano črko: m
Ugibalo se je črko m
Pravilna črka

lo_ omo_ _ _ _
4 / 10 ugib
Dosedanji ugibi ['l', 'o', 'm']
Vnesi ugibano črko: t
Ugibalo se je črko t
Pravilna črka

lo_ omot_ _ _
```

```
Dosedanji ugibi ['l', 'o', 'm', 't']
Vnesi ugibano črko: v
Ugibalo se je črko v
Pravilna črka

lo_ omot_ v_
6 / 10 ugib
Dosedanji ugibi ['l', 'o', 'm', 't', 'v']
Vnesi ugibano črko: i
Ugibalo se je črko i
Pravilna črka

lo_ omotiv_
7 / 10 ugib
Dosedanji ugibi ['l', 'o', 'm', 't', 'v', 'i']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

lo_ omotiva
8 / 10 ugib
Dosedanji ugibi ['l', 'o', 'm', 't', 'v', 'i', 'a']
Vnesi ugibano črko: k
Ugibalo se je črko k
Pravilna črka
ZMAGA
```

Vizualizacija kode

In [36]:

```
# Ustvarimo spremenljivko v katero shranimo besedo katero iščemo
beseda = "lokomotiva"
print(f"Iskana beseda je: {beseda}")

st_ugibov = 10

ugibi = []

won = False

word_check = {}
for ch in beseda:
    word_check[ch] = False

for i in range(st_ugibov):
    print()
    # 1. Izpišemo _ _ _ _ _ _ (in dodamo katere črke so bile uganjene)
    izpis = ""
    for ch in beseda:
        if ch in ugibi:
            izpis += ch
        else:
            izpis += "_ "
    print(izpis)
    print(f"{i+1} / {st_ugibov} ugib")
    print(f"Dosedanji ugibi {ugibi}")

    ugib = input("Vnesi ugibano črko: ")
```

```
    print(f"Ugibalo se je črko {ugib}")
    ugibi.append(ugib)

    # 3. Preverimo ali je črka del besede
    if ugib in beseda:
        print("Pravilna črka")
        word_check[ugib] = True
    else:
        print("Nepravilna črka")

    # 4. Preverimo ali je uganil besedo:
    # 4.a   Če je uganil besedo je zmagal. Konec igre
    # 4.b   Če ni uganil besede in nima več možnih ugibov je izgubil
    # 4.c   Če ni uganil besede in ima še ugibe gremo na korak 1.
    if False in word_check.values():
        won = False
    else:
        won = True
        break



if won:
    print("ZMAGA")
else:
    print("Poraz.")
```

```
Iskana beseda je: lokomotiva


_ _ _ _ _ _ _ _ _ _
1 / 10 ugib
Dosedanji ugibi []
Vnesi ugibano črko: l
Ugibalo se je črko l
Pravilna črka

l_ _ _ _ _ _ _ _ _
2 / 10 ugib
Dosedanji ugibi ['l']
Vnesi ugibano črko: o
Ugibalo se je črko o
Pravilna črka

lo_ o_ o_ _ _ _
3 / 10 ugib
Dosedanji ugibi ['l', 'o']
Vnesi ugibano črko: m
Ugibalo se je črko m
Pravilna črka

lo_ omo_ _ _ _
4 / 10 ugib
Dosedanji ugibi ['l', 'o', 'm']
Vnesi ugibano črko: t
Ugibalo se je črko t
Pravilna črka

lo_ omot_ _ _
5 / 10 ugib
Dosedanji ugibi ['l', 'o', 'm', 't']
Vnesi ugibano črko: v
Ugibalo se je črko v
Pravilna črka

lo_ omot_ v_
6 / 10 ugib
Dosedanji ugibi ['l', 'o', 'm', 't', 'v']
Vnesi ugibano črko: i
Ugibalo se je črko i
Pravilna črka

lo_ omotiv_
7 / 10 ugib
Dosedanji ugibi ['l', 'o', 'm', 't', 'v', 'i']
Vnesi ugibano črko: a
Ugibalo se je črko a
Pravilna črka

lo_ omotiva
8 / 10 ugib
Dosedanji ugibi ['l', 'o', 'm', 't', 'v', 'i', 'a']
Vnesi ugibano črko: k
Ugibalo se je črko k
Pravilna črka
ZMAGA
```

In [ ]:

In [ ]: