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

# 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
```

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 condition is

#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 from a
#Remember to always have a way of exiting the loop! Otherwise it will run endlessly!
```

```
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
```

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
```

Vizualizacija kode

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]
```

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. Vsak korak izpišite 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 upražamo 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}, {u
    elif counter == 7:
        print(f"Dosedanji ugibi: {ugib_0}, {ugib_1}, {ugib_2}, {ugib_3}, {ugib_4}, {u
    elif counter == 8:
        print(f"Dosedanji ugibi: {ugib_0}, {ugib_1}, {ugib_2}, {ugib_3}, {ugib_4}, {u
    elif counter == 9:
        print(f"Dosedanji ugibi: {ugib_0}, {ugib_1}, {ugib_2}, {ugib_3}, {ugib_4}, {u

    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=display)

    # 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 poljubon 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 nista en
```

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.

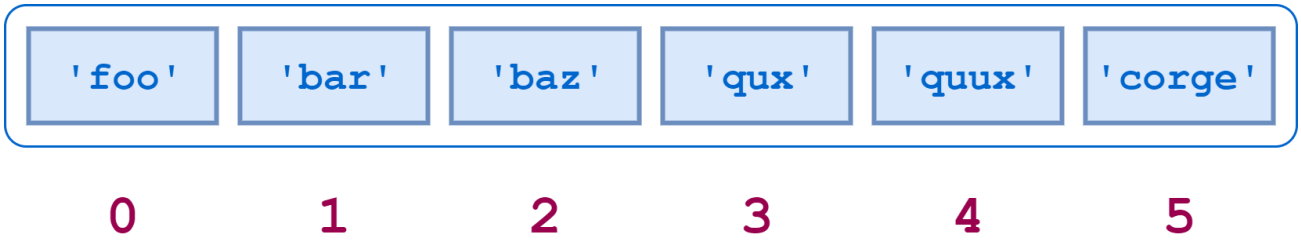
In [12]:
```
a  = ["pingvin", "medved", "los", "volk", "medved"]
a
```

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).

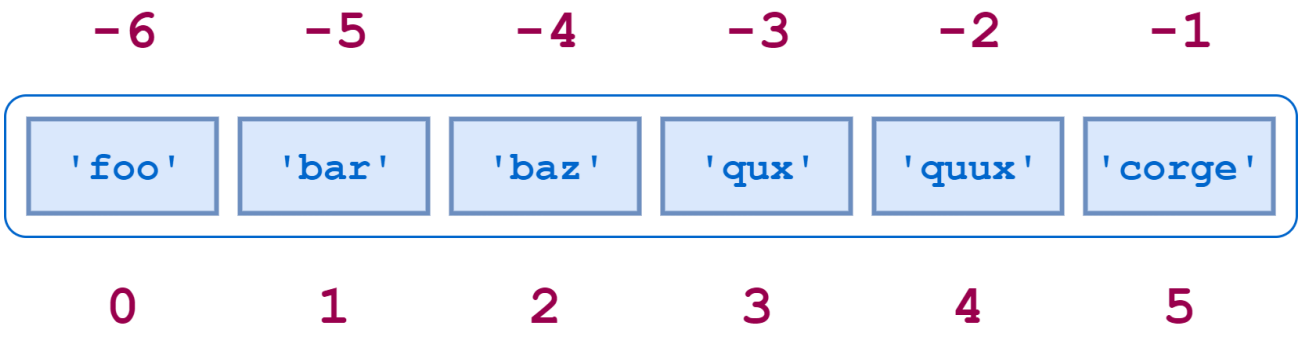


V Pythonu se indexiranje začne z 0.

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 izvzeto
# 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']
```

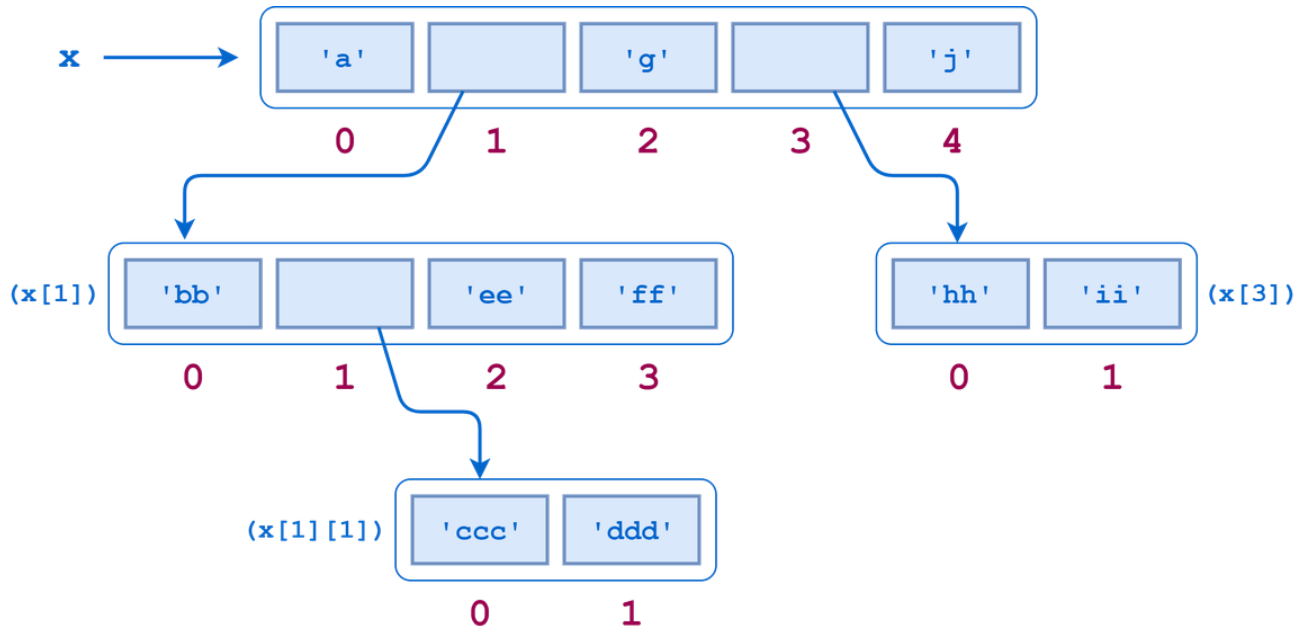
Lists can be nested to arbitrary depth

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.

In [28]:
```
a  = ["pingvin", "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
<class 'int'>
```

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]
```

In [ ]:
```

```