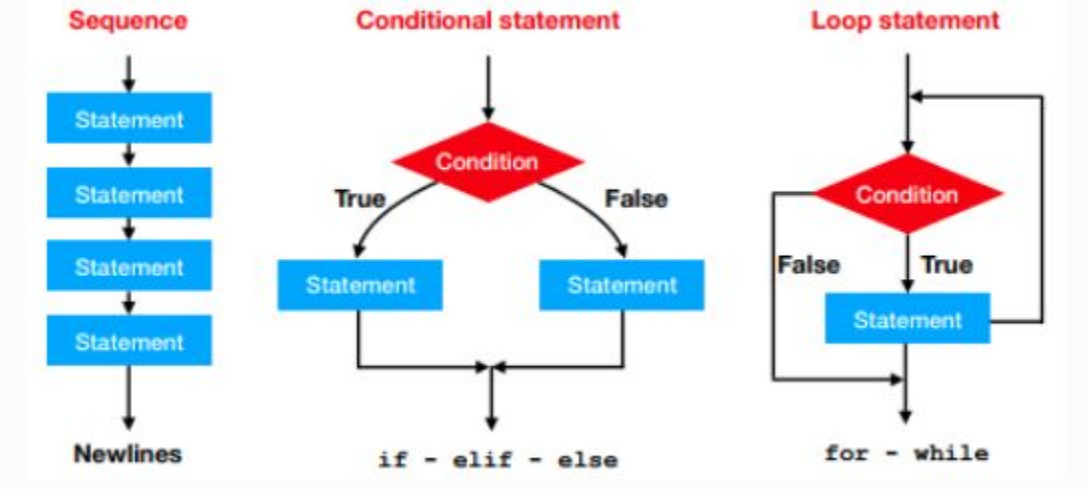
Scientific Programming Practical 5

Introduction

Luca Bianco - Academic Year 2019-20
luca.bianco@fmach.it

More on loops...



Ternary operator

In some cases it is handy to be able to initialize a variable depending on the value of another one.

Example: The discount rate applied to a purchase depends on the amount of the sale. Create a variable *discount* setting its value to 0 if the variable *amount* is lower than 100 euros, to 10% if it is higher.

In [1]:
```
amount = 110
discount = 0

if(amount >100):
    discount = 0.1
else:
    discount = 0 # not necessary

print("Total amount:", amount, "discount:", discount)
```

```
Total amount: 110 discount: 0.1
```

Ternary operator

In some cases it is handy to be able to initialize a variable depending on the value of another one.

Syntax:

```
variable = value if condition else other_value
```

```
amount = 110
discount = 0.1 if amount > 100 else 0
print("Total amount:", amount, "discount:", discount)
```

```
Total amount: 110 discount: 0.1
```

Continue - Break

Sometimes it is useful to **skip an entire iteration** of a loop or **end the loop before its supposed end**.

This can be achieved with two different statements:

continue and **break**.

Continue

Within a **for** or **while** loop, **continue** makes the interpreter skip that iteration and move to the next.

Example: Print all the odd numbers from 1 to 20.

```
#Two equivalent ways
#1. Testing remainder == 1
for i in range(21):
    if(i % 2 == 1):
        print(i, end = " ")

print("")

#2. Skipping if remainder == 0 in for
for i in range(21):
    if(i % 2 == 0):
        continue
    print(i, end = " ")
```

```
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
```

Continue

Continue can be used also within **while** loops but we need to **be careful and remember to update the value of the variable before reaching the continue statement or we will get stuck in never-ending loops.**

Example: Print all the odd numbers from 1 to 20.

```
#Wrong code:
i = 0
while (i < 21):
    if(i % 2 == 0):
        continue
    print(i, end = " ")
    i = i + 1 # NEVER EXECUTED IF i % 2 == 0!!!!
```

Continue

Example: Print all the odd numbers from 1 to 20.

Continue can be used also within **while** loops but we need to **be careful and remember to update the value of the variable before reaching the continue statement or we will get stuck in never-ending loops.**

```
i = -1
while( i< 20):        #i is incremented in the loop, so 20!!!
    i = i + 1         #the variable is updated no matter what
    if(i % 2 == 0 ):
        continue
    print(i, end = " ")
```

```
1 3 5 7 9 11 13 15 17 19
```

Break

Example: Pick a random number from 1 and 50 and count how many times it takes to randomly choose number 27. Limit the number of random picks to 40 (i.e. if more than 40 picks have been done and 27 has not been found exit anyway with a message).

Within a **for** or **while** loop, **break** makes the interpreter exit the loop and continue with the sequential execution. Sometimes it is useful to get out of the loop if to complete our task we do not need to get to the end of the loop.

```
import random

iterations = 1
picks = []
while(iterations <= 40):
    pick = random.randint(1,50)
    picks.append(pick)

    if(pick == 27):
        break
    iterations += 1   #equivalent to iterations = iterations + 1

if(iterations == 41):
    print("Sorry number 27 was never found!")
else:
    print("27 found in ", iterations, "iterations")

print(picks)
```

```
27 found in  14 iterations
[6, 33, 41, 45, 34, 20, 41, 7, 17, 39, 22, 45, 11, 27]
```

Break

Example: Pick a random number from 1 and 50 and count how many times it takes to randomly choose number 27. Limit the number of random picks to 40 (i.e. if more than 40 picks have been done and 27 has not been found exit anyway with a message).

```
import random

iterations = 1
picks = []
while(iterations <= 40):
    pick = random.randint(1,50)
    picks.append(pick)

    if(pick == 27):
        break
    iterations += 1

if(iterations == 41):
    print("Sorry number 27 was never found!")
else:
    print("27 found in ", iterations, "iterations")

print(picks)
```

```
27 found in  14 iterations
[6, 33, 41, 45, 34, 20, 41, 7, 17, 39, 22, 45, 11, 27]
```

```
import random
found = False # This is called flag
iterations = 1
picks = []
while iterations <= 40 and found == False: #the flag is used to exit
    pick = random.randint(1,50)
    picks.append(pick)
    if pick == 27:
        found = True     #update the flag, will exit at next iteration
    iterations += 1

if iterations == 41 and not found:
    print("Sorry number 27 was never found!")
else:
    print("27 found in ", iterations -1, "iterations")

print(picks)
```

```
27 found in  10 iterations
[41, 1, 45, 39, 24, 11, 22, 7, 25, 27]
```

Using breaks or flags….

List comprehension

List comprehension is a quick way to create a list.

The resulting list is normally obtained by applying a function or a method to the elements of another list that **remains unchanged**.

```
new_list = [ some_function (x) for x in start_list if condition]
```

or

if condition optional

```
new_list = [ x.some_method() for x in start_list if condition]
```

List comprehension

Example: Given a list of strings [“dog”, “cat”, “rabbit”, “guinea pig”, “hamster”, “canary”, “goldfish”] create a list with the elements starting with a “c” or “g”.

```
pets = ["dog", "cat", "rabbit", "guinea pig", "hamster", "canary", "goldfish"]

cg_pets = [x for x in pets if x.startswith("c") or x.startswith("g")]

print("Original:")
print(pets)
print("Filtered:")
print(cg_pets)
```

```
Original:
['dog', 'cat', 'rabbit', 'guinea pig', 'hamster', 'canary', 'goldfish']
Filtered:
['cat', 'guinea pig', 'canary', 'goldfish']
```

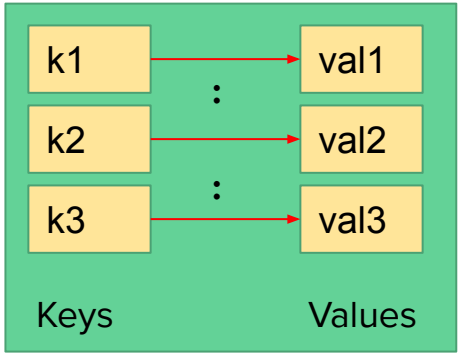
List comprehension

Example: Given the list: ["Hotel", "Icon"," Bus","Train", "Hotel", "Eye", "Rain", "Elephant"] create a list with all the first letters.

```
myList = ["Hotel", "Icon"," Bus","Train", "Hotel", "Eye", "Rain", "Elephant"]
initials = [x[0] for x in myList]

print(myList)
print(initials)
print("".join(initials))
```

```
['Hotel', 'Icon', ' Bus', 'Train', 'Hotel', 'Eye', 'Rain', 'Elephant']
['H', 'I', ' ', 'T', 'H', 'E', 'R', 'E']
HI THERE
```

Dictionaries



A **dictionary** is a map between one object, the **key** and another object, the **value**. Dictionaries are **mutable objects** and contain sequences of mappings *key –> object* but **there is not specific ordering among them.**

Dictionaries are defined using the curly braces **{key1 : value1, key2 : value2}** and **:** to separate keys from values.

```
first_dict = {"one" : 1, "two": 2, "three" : 3, "four" : 4}
print("First:", first_dict)

empty_dict = dict()
print("Empty:",empty_dict)

second_dict = {1 : "one", 2 : "two", "three" :3 } #BAD IDEA BUT POSSIBLE!!!
print(second_dict)

third_dict = dict(zip(["one","two","three","four"],[1,2,3,4]))
print(third_dict)
print(first_dict == third_dict)
```

```
First: {'three': 3, 'one': 1, 'four': 4, 'two': 2}
Empty: {}
{1: 'one', 2: 'two', 'three': 3}
{'three': 3, 'one': 1, 'four': 4, 'two': 2}
True
```

Dictionaries

Keys must be immutable objects

```
a = (1,2,3) #a,b are tuples: hence immutable
b = (1,3,5)

my_dict = {a : 6, b : 9 }
print(my_dict)

c = [1,2,3] #c,d are lists: hence mutable
d = [1,3,5]

dict2 = {c : 6, d : 9}
print(dict2)
```

```
{(1, 3, 5): 9, (1, 2, 3): 6}

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-49-0fe98c7f5acd> in <module>()
      8 d = [1,3,5]
      9
---> 10 dict2 = {c : 6, d : 9}
     11 print(dict2)

TypeError: unhashable type: 'list'
```

Dictionaries

Functions on dictionaries

Result	Operator	Meaning
bool	obj in dict	Return True if a key is present in the dictionary
int	len(dict)	Return the number of elements in the dictionary
obj	dict[obj]	Read the value associate with a key
	dict[obj] = obj	Add or modify the value associated with a key

R/W
mutable!

```
myDict = {"one" : 1, "two" : 2, "twentyfive" : 25}

print(myDict)
myDict["ten"] = 10
myDict["twenty"] = 20
print(myDict)
myDict["ten"] = "10-again"
print(myDict)
print("The dictionary has ", len(myDict), " elements")
print("The value of \"ten\" is:", myDict["ten"])
print("The value of \"two\" is:", myDict["two"])

print("Is \"twentyfive\" in dictionary?", "twentyfive" in myDict)
print("Is \"seven\" in dictionary?", "seven" in myDict)
```

```
{'one': 1, 'two': 2, 'twentyfive': 25}
{'one': 1, 'two': 2, 'twentyfive': 25, 'ten': 10, 'twenty': 20}
{'one': 1, 'two': 2, 'twentyfive': 25, 'ten': '10-again', 'twenty': 20}
The dictionary has  5  elements
The value of "ten" is: 10-again
The value of "two" is: 2
Is "twentyfive" in dictionary? True
Is "seven" in dictionary? False
```

Dictionaries

Methods of dictionaries

Return	Method	Meaning
`list`	`dict.keys()`	Returns the list of the keys that are present in the dictionary
`list`	`dict.values()`	Returns the list of the values that are present in the dictionary
`list of tuples`	`dict.items()`	Returns the list of pairs (key, value) that are present in the dictionary

ERRATUM: `dict.keys()` returns a `dict_keys` object not a list. To cast it to list, we need to call `list(dict.keys())`.

Dictionaries

Accessing a value through the key of a dictionary requires that the pair key-value one searches for is **present** in the dictionary. If the searched key is not present the interpreter crashes out throwing a KeyError

```
myDict = {"one" : 1, "two" : 2, "three" : 3}

print(myDict["one"])
print(myDict["seven"])
```

```
1

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-5-a05b31e54a02> in <module>
      2
      3 print(myDict["one"])
----> 4 print(myDict["seven"])

KeyError: 'seven'
```

Dictionaries

Explicitly test presence of key

```
myDict = {"one" : 1, "two" : 2, "three" : 3}

search_keys = ["one", "seven"]

for s in search_keys:
    if s in myDict:
        print("key:", s, "value:", myDict[s])
    else:
        print("key", s, "not found in dictionary")
```

```
key: one value: 1
key seven not found in dictionary
```

Use get

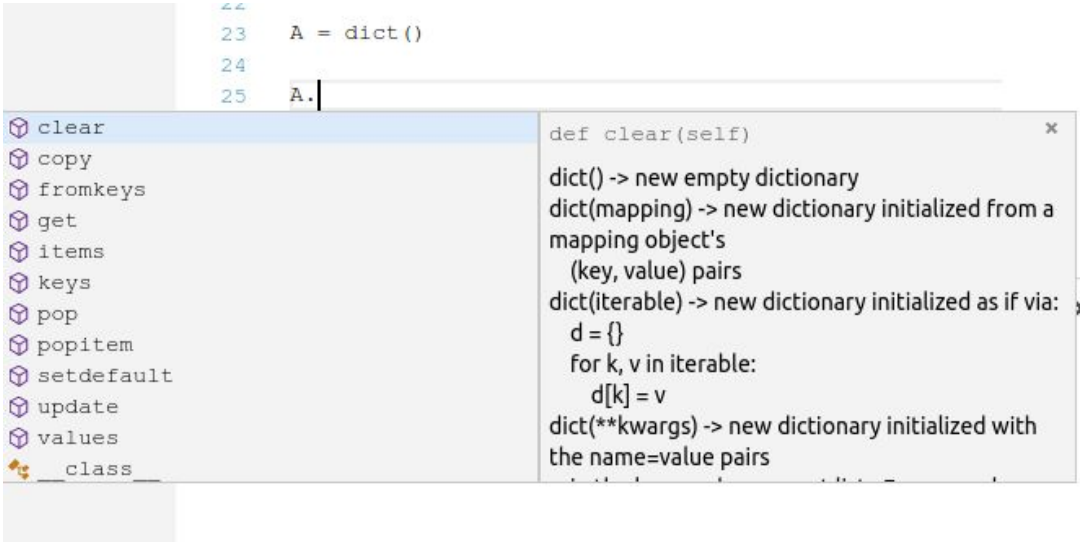
```
myDict = {"one" : 1, "two" : 2, "three" : 3}

search_keys = ["one", "seven"]

for s in search_keys:
    print("key:", s, "value:", myDict.get(s, "not found"))
```

```
key: one value: 1
key: seven value: not found
```

Dictionaries

Return	Method	Meaning
list	dict.keys()	Returns the list of the keys that are present in the dictionary
list	dict.values()	Returns the list of the values that are present in the dictionary
list of tuples	dict.items()	Returns the list of pairs (key, value) that are present in the dictionary

Use the in-line help...



Dictionaries

Example Given the protein sequence below, store in a dictionary all the aminoacids present and count how many times they appear. Finally print out the stats (e.g. how many amino-acids are present, the most frequent, the least frequent and the frequency of all of them **in alphabetical order**).

```
MGNAAAAKKGSEQESVKEFLAKAKEDFLKKWENPAQNTAHL
The number of amino-acids present is  20
A  is present 23 times
C  is present 2 times
D  is present 18 times
E  is present 27 times
F  is present 25 times
G  is present 22 times
H  is present 9 times
I  is present 21 times
K  is present 34 times
L  is present 32 times
M  is present 8 times
N  is present 17 times
P  is present 14 times
Q  is present 14 times
R  is present 15 times
S  is present 16 times
T  is present 14 times
V  is present 20 times
W  is present 6 times
Y  is present 14 times
Amino C has the lowest freq. ( 2 )
Amino K has the highest freq. ( 34 )
```

```
protein = """MGNAAAAKKGSEQESVKEFLAKAKEDFLKKWENPAQNTAHLDQFERIKTLGTGSFGRVML
VKHMETGNHYAMKILDKQKVVKLKQIEHTLNEKRILQAVNFPFLVKLEFSFKDNSNLYMV
MEYVPGGEMFSHLRRIGRFSEPHARFYAAQIVLTFEYLHSLDLIYRDLKPENLLIDQQGY
IQVTDFGFAKRVKGRTWTLCGTPEYLAPEIILSKGYNKAVDWWALGVLIYEMAAGYPPFF
ADQPIQIYEKIVSGKVRFPSHFSSDLKDLLRNLLQVDLTKRFGNLKNGVNDIKNHKWFAT
TDWIAIYQRKVEAPFIPKFKGPGDTSNFDDYEEEEIRVSINEKCGKEFSEF"""

protein = protein.replace("\n","")

print(protein)

amino_acids = dict()

for a in protein:
    if( a in amino_acids):
        amino_acids[a] = amino_acids[a] + 1
    else:
        amino_acids[a] = 1


num_aminos = len(amino_acids)

print("The number of amino-acids present is ", num_aminos)
#let's get all aminoacids
#and sort them alphabetically
a_keys = list(amino_acids.keys())

a_keys.sort()

mostF = {"frequency" : 0, "aminoacid" : "-"}
leastF = {"frequency" : num_aminos, "aminoacid" : "-"}

for a in a_keys:
    freq = amino_acids[a]
    if(mostF["frequency"] < freq):
        mostF["frequency"] = freq
        mostF["aminoacid"] = a

    if(leastF["frequency"] > freq):
        leastF["frequency"] = freq
        leastF["aminoacid"] = a
    print(a, " is present", freq, "times")

print("Amino", leastF["aminoacid"], "has the lowest freq. (",leastF["frequency"],")")
print("Amino", mostF["aminoacid"], "has the highest freq. (",mostF["frequency"],")")
```

http://qcbsciprolab2019.readthedocs.io/en/latest/practical5.html

Exercises

1. Given the following two lists of integers: [1, 13, 22, 7, 43, 81, 77, 12, 15,21, 84,100] and [44,32,7, 100, 81, 13, 1, 21, 71]:
    1. Sort the two lists
    2. Create a third list as intersection of the two lists (i.e. an element is in the intersection if it is present in both lists).
    3. Print the three lists.

Show/Hide Solution

2. Given the string "nOBody Said iT was eAsy, No oNe Ever sald it WoulD be tHis hArd..."
    1. Create a list with all the letters that are capitalized (use str.isupper)
    2. Print the list
    3. Use the string method **join** to concatenate all the letters in a string, using "*" as separator. The syntax of join is str.join(list) and it outputs a string with all the elements in list joined with the character in str (es. "+".join([1,2,3]) returns "1+2+3").

The expected output:

```
['O', 'B', 'S', 'T', 'A', 'N', 'N', 'E', 'I', 'W', 'D', 'H', 'A']
O*B*S*T*A*N*N*E*I*W*D*H*A
```

Show/Hide Solution

3. Given the following list of gene correlations:

```
geneCorr = [["G1C2W9", "G1C2Q7", 0.2], ["G1C2W9", "G1C2Q4", 0.9],
["Q6NMS1", "G1C2W9", 0.8],["G1C2W9", "Q6NMS1",0.4], ["G1C2Q7", "G1C2Q4",0.76]]
```