indented text - aligned part of code (in Python =4 spaces)

```
line of code with no indentation
    indented level 1
       indented level 2
```

(https://docs.python.org/3/tutorial/controlflow.html)

1.CONDITIONS

1A: IF

If the condition is true, then do the indented statements. If the condition is not true, then skip the indented statements.

```
if (condition):
        what to do if condition == True
```

```
name = "Bob"
if (name == "Bob"):
    print ('what a nice name, Bob!') # if name is Bob, this is printed

money = int(input('how much do you have? '))
if (money < 50):
    print ('You are staying at home tonight')
     # this will print only if you have < 50
print ('you have', money) # this is printed everytime
```

1B: IF, ELIF, ELSE

else - another indented block that is only executed when the original condition is false

```
if (condition):
        what to do if condition == True
else:
        what to do if condition != True
```

```
money = input('how much do you have? ')
if (money < 50):
    print ('You are staying at home tonight')
     # this will print only if you have < 50
else:
    print ('a long night') # this will be printed only if you have => 50
print ('you have', money) # this is printed every time
```

elif - adding second condition

```
if (condition1):
        what to do if condition1 == True
elif (condition2):
        what to do if condition1 != True, but condition2 == True
else:
        what to do if both condition1 and condition2 != True
```

```
if (temperature > 50):
    print ('you better stay at home')
elif (temperature < -25):
    print ('you better stay at home')
elif (temperature < 0):
    print ('wear at least a jacket')
else:
    print ('have a nice sunny day)
```

1C: LOGICAL OPERATORS - AND, OR

extending condition

```
if (temperature > 50 or temperature < -25): # saving two lines of code
    print ('you better stay at home')
elif (temperature < 0):
    print ('wear at least a jacket')
else:
    print ('have a nice sunny day)
```

```
have_weapon = True
policeman = False
polite = True

if (have_weapon and !polite):
    print ('a bank is waiting')
if (policeman and !have_weapon):
    print ('get a weapon, quick!')
if (polite and policeman):
    print ('take care near the bank')
```

1D: NESTING CONDITIONS

```
feeling_sick = True
money = 150
car = False

if (feeling_sick):
    if (car):
        print ('drive to the hospital')
        elif (money > 100000)
           print ('take taxi, whatever')
        else:
           print ('take a bus')
    else:
        if (car and money > 100)
           print ('plan a trip somewhere')
        else:
           print ('at least you are not sick')
```

2.LISTS

Data structure type for storing sequence of values. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.

2A:DEFINING LIST

Values in list are inserted inside **[ ]** brackets, separated by coma **,**

```
#list with only string values
should_buy = ['carrot', 'milk', 'eggs', 'yogurt']

#list with only numbers
scores = [12, 15, 5, 13, 42, 13]

#list with mixed values types
a_mix = ['elephant', 'jacket', 154, True, 'R602', 0.00789]
```

2B: ACCESSING VALUES IN LIST

Use the square brackets with the index or indices to obtain value available at that index

```
        activities = ['sleeping', 'eating',  'commuting', 'working', 'meeting', 
'working', 'commuting', 'showering', 'sleeping']
        print (activities[0] ) # 'sleeping' -> returns string
        print (activities[2:4]) # ['commuting', 'working'] -> returns list
        print (activities[-2]) # 'showering'
```

2B: UPDATING VALUES IN LIST

updating value in position by **[] notation**

```
my_favourite_animals = ['raindeer', 'frog', 'sheep', 'tiger']
my_favourite_animals[1] = 'dog'
print (my_favourite_animals) # ['raindeer', 'dog', 'sheep', 'tiger']

my_favourite_animals[1:3] = ['cat', 'camel']
print (my_favourite_animals) # ['raindeer', 'cat', 'came', 'tiger']
```

insert(index, item) - put item on selected index and move other items. Insert is not removing anything

```
        todo_list = ['buy eggs', 'buy milk', 'go home', 'prepare pancakes']
        todo_list.insert(1, 'buy oil')
        # todo_list is now  ['buy eggs', 'buy oil', 'buy milk', 'go home', 'prepare
pancakes']
```

append(item) - add an item to the end of the list

```
        items_on_my_desk = ['pen', 'paper', 'coffee']
        items_on_my_desk.append ('another paper')
        # items_on_my_desk is now ['pen', 'paper', 'coffee', 'another paper']
```

2C: DELETING VALUES IN LIST

del - removes value at index

```
        my_friends = ['Bob', 'David', 'John', 'Alice']
        del my_friends[1]
        print(my_friends) # ['Bob', 'John', 'Alice']

        my_friends = ['Bob', 'David', 'John', 'Alice']
        del my_friends[1:3]
        print(my_friends) # ['Bob', 'Alice']

        my_friends = ['Bob', 'David', 'John', 'Alice']
        del my_friends[1:]
        print(my_friends) # ['Bob']
```

remove() - removes first occurrence of selected value/object

```
        my_friends = ['Bob', 'David', 'John', ‘Alice’]
        my_friends.remove('David')
        print(my_friends) # ['Bob', 'John', 'Alice']

        busses = [21, 15, 13, 15, 23, 10, 5, 4, 21]
        busses.remove(21)
        print(busses) # [15, 13, 15, 23, 10, 5, 4, 21]
        # only first occurrence of 21 is deleted
```

2D: OPERATIONS WITH LIST

len() returns number of values/length of list

```
my_friends = ['Bob', 'David', 'John', 'Alice']
print('I have', len(my_friends), 'friends') # 4 friends
```

concatenation with **+** - merging more lists

```
my_vegetables = ['carrot', 'potato']
your_vegetables = ['tomato']
our_vegetables = my_vegetables  + your_vegetables + ['spinach']
# our_vegetable is now ['carrot', 'potato', 'tomato', 'spinach']
```

in returns True if item is in list

```
print ('green' in ['blue', 'red', 'yellow']) # False
print (25 in [1,5,36,25]) # True
```

max(), **min()**

```
regions_population = [1562, 45847, 15454, 1521, 48572, 876, 6797]
print(max(regions_population)) # 48572
print(max(regions_population)) # 876
```

2E: LIST METHODS

append(), insert() - see 2B - updating values in list

count() - number of times x appears in the list

```
chars = ['a', 'b', 'c', 'b', 'a', 'a', 'b', 'c']
print (chars.count('b')) #3
```

index() - position of item in list (index 0 = position 1)

```
places = ['Ghana', 'Togo', 'Niger', 'Liberia']
print(places.index('Togo')) # 1
```

reverse()

```
chars = ['a', 'b', 'c', 'd', 'e']
chars.reverse()
print(chars) # ['e', 'd', 'c', 'b', 'a']
```

2F: LIST CONVERSION

split() - converts string to list. First parameter is a separator (default value -space char)

```
a_text = 'Prague, Munchen, Vienna, Berlin, Skopje'
a_list = a_text.split(', ')
```

```
# a_list is now ['Prague', 'Munchen', 'Vienna', 'Berlin', 'Skopje']
```

join() - converts list to string. First parameter is joining character/s

- join() is a method of string data type, not list data type.

```
route = ['Brno', 'Breclav', 'Bratislava', 'Gyor', 'Budapest']
print ('recommended route is :' ' - '.join(route))
# recommended route is : Brno - Breclav - Bratislava - Gyor - Budapest
```

2F: LIST INSIDE LIST

nested list structure, used quite often in geoinformatics (coordinates, rasters, ...)
parking slots example

```
# parking area is just a line of parking places
paring_area = ['car1', False, ' False, 'car2', 'car3'] # 1* 5 parking places
parking_area[2] # accesing car with index 2

# car park has more "lines", is "2D"
car_park = [ ['car1'. False, False], ['car2', 'car3', 'car4'], ['car5',
False, 'car6'] ] # 3 * 3 parking places
parking _area[0][2] # accesing car with index 2 in "line" 1

# multi-storey parking building with "3D" parking - floor and "line"
parking_building = [ [ ['car1', False], ['car2', 'car3'] ], [ [False, False],
[False, 'car4'] ], [ ['car5', False], ['car6', 'car7'] ] ] # three floors,
each with 2 lines with 2 parking slots
parking_building[1][0][1] # getting parking slot on the 2. floor, 1. line and
2. position
```

3.CYCLES

3A: FOR CYCLE

Basic iteration - repeating nested code for every element in definition
Iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

cycling through string value:

```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
for letter in alphabet:
	# variable letter is declared to be one character of alphabet in every
iteration
	# number of iterations is equal to len(alphabet)
```

```
            print(letter)
```

cycling through list

```
invited_people = ['Bob', 'Alan', 'Emily', 'Gustav']
for invited_person in invited_people:
      print ('send invitation letter to', invited_person)
      # this will print:
      # 'send invitation letter to Bob'
      # 'send invitation letter to Alan'
      # 'send invitation letter to Emily'
      # 'send invitation letter to Gustav'
```

using **range()** function to create incrementing numbers

```
for vertex in range(0,3):
    print(vertex)
    # will print
    # 0
    # 1
    # 2
```

range() takes one to three arguments:

```
range(5) # -> 0, 1, 2, 3, 4
range(2,8) # -> 2, 3, 4, 5, 6, 7
range(4,18,3) # -> 4, 7, 10, 13, 16
```

condition inside cycle example:

```
edges_in_polygon = [2,4,7,3,5,9,4,1,6]
for polygon in edges_in_polygon:
    if polygon == 3:
      print ('this is a triangle')
    elif polygon > 10:
      print ('complex polygon, need to generalize')
```

creating a new list inside cycle

```
# A_party is only for people with name beginning with 'A' character
people = ['Alice', 'Antonny', 'Bob', 'Cecilia', 'Bernard', 'David', 'Alex']
people_invited = []

for person in people:
    if person[0] == 'A':
      people_invited.append(person)
```

```
print(people_invited)
```

another examples

```
# 1. calculating inside cycle
values = [4,5,9,14,25,3,12]

# getting sum of values
sum = 0
for value in values:
    sum += value

# getting maximum of values
max = 0
for value in values:
    if value > max:
        max = value

# 2. cycling through indexes with range() to get values from more lists
# in case that more lists have same length and are related by index
students_id = [17, 56, 123, 4578, 265, 5]
students_points = [4, 7, 2, 12, 5, 13]

for i in range(0, len(students_id) - 1):
    # range(0, len(students_id) - 1) creates set of indexes inside student_id
list
    print ('student with id' , students_id[i] , 'has', students_points[i],
'points')
```

cycle nested in cycle:

```
chars1 = ["a", "b", "c"]
chars2 = ["d", "e", "f"]

for i in chars1:
    for j in chars2:
       print (i + j)

# prints: ad, ae, af, bd, be, bf, cd, ce, cf


# looking for position of "car2"
parking_slots = [[False, False, "car1"], [False, "car3", False], ["car2",
False]]

for p in range(len(parking_slots)):
    for q in range(len(parking_slots[p])):
       if (parking_slots[p][q] == "car2"):
```

```
print ("car3 position: ", p, q)
```

3A: WHILE

the *while* loop will run until a defined condition is met

examples:

```
# changing value until it meets conditions
count = 5

while count < 100:
    count += 30
    print('count is changed to', count)

print('count is now bigger than 100, it is', count)
# ‘count is changed to 35’
# ‘count is changed to 65’
# ‘count is changed to 95’
# ‘count is changed to 125’
# ‘count is now bigger than 100, it is 125’


# repeating input
course_ranking = input('what do you think about this course?' )

while (course_ranking != 'perfect'):
    course_ranking = input('what do you think about this course?' )

print ('thank you for your honest opinion')
```

notice :

- while is in general redundant, it is possible to change it to if condition
- take care about infinite loops

```
# this is a bad idea
i = 5
while i < 6:
    print (i)
```