

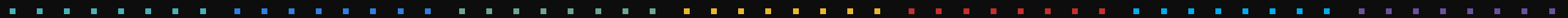
WMNBE-2203 | BACK-END DEVELOPMENT

Python #3

Control | Sequences | Functions



ΠΕΡΙΕΧΟΜΕΝΑ

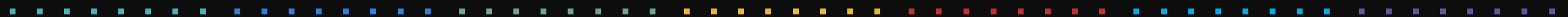


Περιεχόμενα

- Control statements
 - Conditionals
 - Loops
- Sequences
 - **list**
 - **tuple**
 - **dict**
- Functions



CONTROL STATEMENTS



Control Statements

off-side rule

Όπως έχει ήδη αναφερθεί, η **Python** χρησιμοποιεί τον κανόνα **off-side**. Άρα, για τη σύνταξη των **blocks** λαμβάνεται υπόψη το **indentation**, η στοίχιση των εντολών.

Αντί για τη χρήση ειδικών χαρακτήρων (π.χ. curly brackets) ή συγκεκριμένων keywords (π.χ. begin, end) ο ορισμός των **blocks** γίνεται με τα κενά στην αρχή της γραμμής.

Δύο διαδοχικές γραμμές που είναι "στοιχισμένες" βρίσκονται στο ίδιο **block**. Μία εντολή που βρίσκεται στοιχισμένη πιο δεξιά από την προηγούμενή της, ανήκει στο **block** αυτής της εντολής.

Control Statements

if...elif...else

Η βασική εντολή επιλογής/απόφασης στην **Python** είναι η εντολή **if**, σε οποιαδήποτε μορφή της.

Από την έκδοση **3.10** η **Python** διαθέτει και την εντολή **match**, η οποία θυμίζει την εντολή **switch** άλλων γλωσσών προγραμματισμού, αλλά διαφέρει αρκετά.

Σύνταξη

if...elif...else

```
if <condition_1>:
    <action_1>
elif <condition_2>:
    <action_2>
...
else:
    <action_n>
```

Τα τμήματα **elif** και **else** είναι προαιρετικά. Μπορούν να υπάρχουν περισσότερα από ένα τμήματα **elif**.

if...elif...else

Λειτουργία

Αν ισχύει η συνθήκη (λογική έκφραση) στην πρώτη γραμμή της **if** εκτελούνται οι εντολές του πρώτου **block**.

Διαφορετικά, εξετάζονται (αν υπάρχουν) οι επόμενες συνθήκες διαδοχικά και εκτελούνται οι εντολές του πρώτου (στη σειρά) **block** για το οποίο ικανοποιείται η αντίστοιχη συνθήκη.

Αν καμία συνθήκη δεν ικανοποιείται, εκτελούνται οι εντολές του **block else** (αν υπάρχει).

Παράδειγμα #1

if...elif...else

```
username = input('Enter your username: ')

if username == 'admin':
    print('Welcome admin!')
```

Παράδειγμα #2

if...elif...else

```
age = int(input('Enter your age: '))

if age >= 18:
    print('You may enter!')
else:
    print('Please leave.')
```

Παράδειγμα #2

if...elif...else

```
grade = int(input('Enter your grade: '))

if grade >= 18:
    print('Congrats!')
elif grade >= 16:
    print('You did well')
elif grade >= 14:
    print('Good for you')
elif grade >= 10:
    print('You should try a bit harder!')
else:
    print('You shall not pass!')
```

Shorthands

if...elif...else

Αν έχουμε μία μόνο εντολή στο **block**

```
if x > 0: print('Θετικός αριθμός')
```

Αν έχουμε και τμήμα **else**

```
print('Θετικός') if x > 0 else print('Μη θετικός')

# Conditional expression (Ternary operator?)
print('Θετικός' if x > 0 else 'Μη θετικός')
```

Σύνταξη

match...case

```
match subject:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    ...
    case _:
        <action_wildcard>
```

Το τμήμα **case _** είναι προαιρετικό.

match...case

Λειτουργία

Η εντολή **match** δέχεται μια έκφραση και συγκρίνει την τιμή της, τον τύπο της και το σχήμα της, με διαδοχικά "μοτίβα" (**patterns**) που ορίζονται σε μία ή περισσότερες δηλώσεις **case** (*structural mapping*).

Η σύγκριση γίνεται από πάνω προς τα κάτω, μέχρι να βρεθεί κάποια αντιστοίχιση. Αν βρεθεί κάποια αντιστοίχιση, εκτελείται η ενέργεια **action** που σχετίζεται με το μοτίβο της.

Εάν δεν επιβεβαιωθεί μια ακριβής αντιστοίχιση, η τελευταία περίπτωση **case _** (εάν παρέχεται), θα χρησιμοποιηθεί ως περίπτωση αντιστοίχισης.

Εάν δεν επιβεβαιωθεί μια ακριβής αντιστοίχιση και δεν υπάρχει περίπτωση "μπαλαντέρ", ολόκληρο το μπλοκ αντιστοίχισης είναι **no-op**.

Παράδειγμα #1

match...case

```
match status:
    case 400:
        return "Bad request"
    case 404:
        return "Not found"
    case 418:
        return "I'm a teapot"
    case 401 | 403 | 404:
        return "Not allowed"
    case _:
        return "Something's wrong with the internet"
```


Παράδειγμα #2

match...case

```
match point:
    case (0, 0):
        print("Origin")
    case (0, y):
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")
    case _:
        raise ValueError("Not a point")
```

Παράδειγμα #3

match...case

```
class Point:
    x: int
    y: int

def location(point):
    match point:
        case Point(x=0, y=0):
            print("Origin is the point's location.")
        case Point(x=0, y=y):
            print(f"Y={y} and the point is on the y-axis.")
        case Point(x=x, y=0):
            print(f"X={x} and the point is on the x-axis.")
        case Point():
            print("The point is located somewhere else on the plane.")
        case _:
            print("Not a point")
```

Παράδειγμα #4

match...case

```
match points:
    case []:
        print("No points in the list.")
    case [Point(0, 0)]:
        print("The origin is the only point in the list.")
    case [Point(x, y)]:
        print(f"A single point {x}, {y} is in the list.")
    case [Point(0, y1), Point(0, y2)]:
        print(f"Two points on the Y axis at {y1}, {y2} are in the list.")
    case _:
        print("Something else is found in the list.")
```

Παράδειγμα #5

match...case

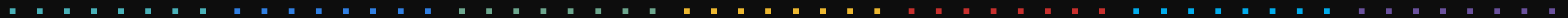
```
match test_variable:
    case ('warning', code, 40):
        print("A warning has been received.")
    case ('error', code, _):
        print(f"An error {code} occurred.")
```

Παράδειγμα #6

match...case

```
# Using an if clause as a guard
match point:
    case Point(x, y) if x == y:
        print(f"The point is located on the diagonal Y=X at {x}.")
    case Point(x, y):
        print(f"Point is not on the diagonal.")
```

LOOPS



Loops

while...else

Η λειτουργία της εντολής **while** στην **Python** είναι η αναμενόμενη... but with a small twist.

Σύνταξη

```
while <condition>:
    <statements>
else:
    <statements>
```

while...else

Λειτουργία

Όσο ισχύει η συνθήκη (λογική έκφραση) εκτελούνται οι εντολές στο **block** της **while**.

Αν *δεν διακοπεί* η επαναλαμβανόμενη λειτουργία της **while** με άλλο τρόπο (πέραν της συνθήκης), θα εκτελεστούν στο τέλος και οι εντολές του **block else**.

Παράδειγμα

while...else

Εμφάνισε τους αριθμούς από το 1 μέχρι και το 10:

```
i = 1
while i <= 10:
    print(i, end=' ')
    i += 1
else:
    print('\n')
```

while...else

Εντολές **break** & **continue**

Η μόνη περίπτωση να διακοπεί η λειτουργία μιας **while**, πριν η συνθήκη της γίνει **False**, είναι να υπάρχει στο σώμα της μια εντολή **break**.

Η εντολή **break** διακόπτει τη λειτουργία της πιο "κοντινής" **while** (ή **for**). Επίσης, δεν ενεργοποιείται το τμήμα **else**.

Η εντολή **continue** μεταφέρει άμεσα τον έλεγχο πίσω στη γραμμή της **while**. Αν ισχύει ακόμα η συνθήκη της ξεκινά η "επόμενη" επανάληψη.

Παράδειγμα #1

while...else

Αναζήτηση του χαρακτήρα **ñ** μέσα σε μια συμβολοσειρά:

```
i = 0
s = 'My name is Iñigo Montoya'
while i < len(s):
    if (s[i] == 'ñ'):
        print('Found at pos:', i)
        break;
    i += 1
else:
    print('Not found')
```

while...else

Παράδειγμα #2

Εμφάνισε τους αριθμούς από το 1 μέχρι και το 100, εκτός των πολλαπλάσιων του 7:

```
i = 0
while i < 100:
    i += 1
    if (i % 7 == 0): continue
    print(i)
```

while...else

Assignment expressions / the walrus operator :=

Ο τελεστής *walrus* κάνει ανάθεση τιμής και επιστρέφει παράλληλα την τιμή αυτή, ως έκφραση.

Η χρήση του μας οδηγεί, σε αρκετές περιπτώσεις, σε αρκετά πιο συνοπτικό κώδικα.

```
list = []
while (x := int(input('Enter number: '))) != 0:
    if x < 0:
        continue
    list += [x]      # l.append(x)
print(list)
```

Loops

for...else

Η εντολή **for** στην **Python** είναι διαφορετική από μία κλασική **for** μιας κάποιας άλλης γλώσσας.

Θυμίζει περισσότερο τις εντολές / συναρτήσεις **foreach**, **for...in**, **for...of**, **forEach** άλλων γλωσσών.

Και εδώ, όπως και στην **while**, υπάρχει το τμήμα **else**, με την ίδια λειτουργία.

Σύνταξη

for...else

```
for <variable> in <sequence>:  
    <statements>  
else:  
    <statements>
```

, όπου **sequence** μια ακολουθία στοιχείων, όπως **range**, **list**, **tuple**, **dict**^{*}, **set**, **string**.

for...else

Λειτουργία

Εκτελείται μια επανάληψη για κάθε ένα από τα στοιχεία της δεδομένης ακολουθίας στοιχείων.

Το στοιχείο αυτό είναι διαθέσιμο μέσω της μεταβλητής που δηλώθηκε.

Αν η εντολή **for** δεν διακοπεί πρόωρα (μέσω μιας **break**), θα εκτελεστεί και το τμήμα **else**.

for...else

Παράδειγμα #1

Εμφάνισε τα γράμματα μίας λέξης και τις λέξεις μιας φράσης:

```
for char in 'Montoya':
    print(char)

for word in 'My name is Iñigo Montoya'.split(' '):
    print(word)
```

for...else

Παράδειγμα #2

Αναζήτηση του χαρακτήρα **ñ** μέσα σε μια συμβολοσειρά:

```
for char in 'My name is Iñigo Montoya':
    if char == 'ñ':
        print('Found!')
        break
else:
    print('Not found...')
```

for...else

Συνάρτηση **range**

Για να λειτουργήσει η **for** με έναν πιο "παραδοσιακό" τρόπο, δηλαδή με μια ακολουθία ακέραιων αριθμών, μπορεί να χρησιμοποιηθεί η συνάρτηση **range**, η οποία παράγει την απαιτούμενη ακολουθία.

for...else

Συνάρτηση **range**

Αν δοθεί ένα όρισμα *stop*, τότε παράγεται μια ακολουθία αριθμών από το *0* μέχρι το *stop-1*.

```
# 10 iterations
# prints numbers between 0 and 9

for i in range(10):
    print(i)
```

for...else

Συνάρτηση **range**

Αν δοθούν δύο ορίσματα *start* & *stop*, τότε παράγεται μια ακολουθία αριθμών από το *start* μέχρι το *stop-1*.

```
# 5 iterations (10 - 5)
# prints numbers between 5 and 9

for i in range(5, 10):
    print(i)
```

for...else

Συνάρτηση **range**

Αν δοθεί τιμή και για το όρισμα *step*, τότε δύο διαδοχικοί αριθμοί της ακολουθίας διαφέρουν κατά *step*.

```
# prints numbers 10, 13, 16, 19
for i in range(10, 20, 3):
    print(i)
```

Όλα τα ορίσματα μπορεί να είναι και αρνητικοί ακέραιοι, αριθμοί. Αν το όρισμα *step* δε "συμφωνεί" με τα *start* και *stop*, η ακολουθία θα είναι κενή.

Η τιμή *0* για το όρισμα *step* δεν είναι αποδεκτή.

Sequences

list

Στην **Python** δεν υπάρχουν πίνακες. Η δομή που προσφέρει η **Python** στη θέση τους είναι η λίστα (**list**).

Η λίστα είναι μία δυναμική δομή δεδομένων και αναπαριστά μια ακολουθία διατεταγμένων στοιχείων, τα οποία μπορούν να προσπελαστούν μέσω κάποιου δείκτη.

Τα στοιχεία δεν είναι απαραίτητα του ίδιου τύπου.

Δήλωση / Αρχικοποίηση

list

```
>>> l1 = []                # empty list
>>> l1 = list()           # empty list
>>> l3 = [1, 2, 3]
>>> l3
[1, 2, 3]
>>> l4 = list('Python')
>>> l4
['P', 'y', 't', 'h', 'o', 'n']
```


Προσπέλαση

list

```
>>> l = [1, 2, 4]
>>> l[0]
1
>>> l[2] = 3
>>> l
[1, 2, 3]
```

list

Τελεστές

* για επανάληψη στοιχείου / στοιχείων

```
>>> l = [0] * 5
>>> l
[0, 0, 0, 0, 0]
```

+ (+=) για προσάρτηση στοιχείων

```
>>> l = [1, 2]
>>> l += [3, 4]
>>> l
[1, 2, 3, 4]
```

list

Μήκος λίστας / πλήθος στοιχείων

```
>>> l = [1, 2]
>>> len(l)
2
>>> l += [3]
>>> len(l)
3
```

list

Slice Notation

Η **Python** έχει ένα μοναδικό τρόπο για να αναφερθεί σε κομμάτι μίας λίστας.

```
list[start:stop:end]
```

Επιστρέφει τμήμα της λίστας, το οποίο ξεκινά από τη θέση *start* και σταματά στο στοιχείο **πριν** τη θέση *stop*, με βήμα *step*.

Η σύνταξη αυτή μπορεί να χρησιμοποιηθεί είτε για ανάκτηση, είτε, ακόμα, και για "αντικατάσταση" ενός τμήματος λίστας.

Παράδειγμα

list

```
>>> l = [1, 2, 3, 4, 5]
>>> l[1:]           # all but first
[2, 3, 4, 5]
>>> l[:len(l)-1]    # all but last
[1, 2, 3, 4]
>>> l[:-1]          # all but last
[1, 2, 3, 4]
>>> l[1:4]
[2, 3, 4]
>>> l[::-1]         # reverse
[5, 4, 3, 2, 1]
>>> copy = l[:]     # clone
```

list

Συναρτήσεις

Συνάρτηση Λειτουργία

.append(x)	προσθήκη του x στο τέλος της λίστας	a[len(a):] = [x]
.extend(l)	προσάρτηση των στοιχείων της λίστας l	a[len(a):] = l
.insert(i,x)	εισαγωγή του x στη θέση i	
.remove(x)	διαγραφή της πρώτης εμφάνισης του x	πιθανό ValueError
.pop(i)	αφαίρεση/ανάκτηση από τη θέση i	default i=len(a)-1
.index(x)	η θέση του στοιχείου x	πιθανό ValueError
.count(x)	πόσες φορές εμφανίζεται το x	
.sort()	ταξινόμηση των στοιχείων	
.reverse()	αντιστροφή των στοιχείων	

Sequences

tuple

Εναλλακτικά της λίστας, μπορεί να χρησιμοποιηθεί μία πλειάδα.

Η πλειάδα χρησιμοποιείται για αμετάβλητες ακολουθίες συγκεκριμένου μήκους, καθώς η πλειάδα είναι μία *immutable* δομή δεδομένων, όπως και η συμβολοσειρά.

Ως εκ τούτου, μετά τη δήλωσή τους, δεν μπορούν να προστεθούν, αφαιρεθούν ή αντικατασταθούν στοιχεία.

tuple

Δήλωση

Η δήλωση τους γίνεται με τη χρήση των παρενθέσεων, ή και χωρίς αυτές, και με την παράθεση των στοιχείων χωρισμένα με κόμμα.

```
t1 = (1, 2)
t2 = 3, 4
print(type(t1), type(t2)) # <class 'tuple'> <class 'tuple'>
```


tuple

Προσπέλαση

Η ανάκτηση των δεδομένων γίνεται όμοια με τις λίστες. Τροποποίηση, όπως ειπώθηκε, δεν μπορεί να γίνει.

```
>>> t = ('Monty', 'Python', 'and', 'the', 'Holy', 'Grail')
>>> t[0]
Monty
>>> t[4:]
('Holy', 'Grail')
>>> a, b = t[:2]
>>> a
Monty
>>> b
Python
```

tuple

Αντιμετάθεση

Η χρήση των **tuple** μας δίνει και έναν πρωτότυπο τρόπο για να κάνουμε αντιμετάθεση δύο μεταβλητών, *the Pythonic way*.

```
>>> x = 7
>>> y = 4
>>> x, y = y, x
>>> x
4
>>> y
7
```

tuple

enumerate

Σε μία τυπική χρήση της εντολής **for** στην **Python**, δεν έχουμε κάποιον μετρητή ώστε να γνωρίζουμε σε ποια επανάληψη (*iteration*) βρισκόμαστε.

Αυτό μπορεί να λυθεί με τη συνάρτηση **enumerate**.

Η **enumerate** δέχεται μία ακολουθία στοιχείων και επιστρέφει μία ακολουθία πλειάδων, όπου το πρώτο στοιχείο (κάθε πλειάδας) είναι το *index* (το οποίο ξεκινά από την τιμή *start*) και το δεύτερο το αρχικό στοιχείο της ακολουθίας.

```
enumerate(seq, start)
```

tuple

enumerate

```
>>> list(enumerate(list('Python')))
[(0, 'P'), (1, 'y'), (2, 't'), (3, 'h'), (4, 'o'), (5, 'n')]
```

```
grades = [17, 18, 20, 19]
for i, x in enumerate(grades, 1):
    print(f'Grade {i}: {x}')
```

Sequences

dict

Μία ακόμα, πολύ χρήσιμη, δομή δεδομένων είναι τα λεξικά (*dictionaries*). Στα λεξικά αποθηκεύουμε *key-value pairs*.

Σε αντίθεση με τις λίστες, η αναφορά στα στοιχεία ενός λεξικού γίνεται με ένα *κλειδί* και όχι με τη θέση.

Παλαιότερα τα λεξικά δεν αποτελούνται από διατεταγμένα στοιχεία, αλλά αυτό άλλαξε μετά την έκδοση 3.6.

Αυτό σημαίνει πως μπορώ να ανακτήσω τα περιεχόμενα του λεξικού σειριακά, γνωρίζοντας ότι αυτά διατηρούν τη σειρά με την οποία δηλώθηκαν / τοποθετήθηκαν.

Δήλωση / Αρχικοποίηση

dict

```
>>> d1 = {}                # empty dict
>>> d2 = dict()           # empty dict
>>> d3 = {
    'brand': 'Tesla',
    'model': 'Model S',
    '0-60': 2.1
}
```

Προσπέλαση

dict

```
...
>>> len(d3)
3
>>> d3['brand']
Tesla
>>> for key in d3:
        print(key, d3[key])
brand Tesla
model Model S
0-60 2.1
```

Προσοχή: όταν χρησιμοποιείται **dict** σε εντολή **for**, οι τιμές που παίρνει η μεταβλητή της **for** προκύπτουν από τα **keys** και όχι από τα **values**.

dict

Μέθοδος `.items()`

Ανάκτηση των στοιχείων του λεξικού ως πλειάδες.

```
...
>>> for key, value in d3.items():
        print(key, value)
brand Tesla
model Model S
0-60 2.1
```


dict

Μέθοδος **.keys()**

Ανάκτηση μόνο των κλειδιών.

```
...
>>> for key in d3.keys():
        print(key, value)

brand
model
0-60
```

Σημείωση: Σε μία **for** η χρήση της μεθόδου **.keys()** δεν είναι ιδιαίτερα χρήσιμη, καθώς και χωρίς αυτή, η μεταβλητή της **for** θα πάρει τιμές από τα κλειδιά του λεξικού.

dict

Μέθοδος `.values()`

Ανάκτηση μόνο των τιμών.

```
...
>>> for value in d3.values():
        print(key, value)
Tesla
Model S
2.1
```

Sequences

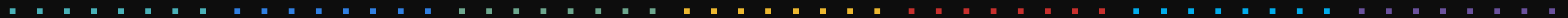
Τελεστής **in**

Ένας χρήσιμος τελεστής για όλες τις προαναφερθείσες δομές δεδομένων είναι ο τελεστής **in**.

Εξετάζει αν υπάρχει ή όχι ένα στοιχείο στη δομή και έχεις ως αποτέλεσμα **True** / **False**, αντίστοιχα.

```
>>> 3 in [1, 2, 3]
True
>>> 6 in (4, 5)
False
>>> '001' in { '001': 'Python' }
True
>>> 'Python' in { '001': 'Python' }
False
```

FUNCTIONS



Functions

Ορισμός / Κλήση

Για τον ορισμό *custom* συναρτήσεων, στην **Python**, χρησιμοποιούμε το *keyword* **def**. Το σώμα της συνάρτησης ορίζεται, όπως και κάθε *block* στην **Python**, μέσω του *indentation*.

Η κλήση γίνεται απλά με το όνομα της συνάρτησης, συνοδευόμενο από παρενθέσεις.

```
def hello_world():  
    print('Hello world!')  
  
hello_world()  
hello_world()
```

Functions

Πέρασμα τιμών

Το πέρασμα τιμών σε μια συνάρτηση γίνεται μέσω παραμέτρων.

Οι παράμετροι αυτές ορίζονται μέσα στις παρενθέσεις στη δήλωση της συνάρτησης και μπορεί να είναι και προαιρετικές (*optional*), αρκεί να τους ορίσουμε προεπιλεγμένες (*default*) τιμές.

Για την επιστροφή τιμής (ή και τιμών), από τη συνάρτηση, χρησιμοποιείται η εντολή **return**.

```
def min(a, b=0):  
    return a if a < b else b
```

Functions

Παράδειγμα #1

```
def minmax(list):
    min = max = list[0]

    for x in list[1:]:
        if x < min:
            min = x
        if x > max:
            max = x

    return min, max

res = minmax([3, 7, 4, 9, 6])
print(res, type(res))
```

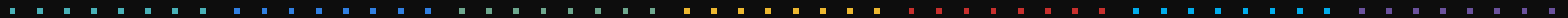
Functions

Παράδειγμα #2

```
def greet(name, msg="Hello"):
    print(f'{msg} {name}!')

greet('Alice')           # Hello Alice!
greet('Bob', 'Howdy')    # Howdy Bob!
```


HOMework



Homework

Homework 1

Γράψτε ένα πρόγραμμα σε **Python**, το οποίο να ζητά τρεις αριθμούς και να τους εμφανίζει ταξινομημένους από το μικρότερο στο μεγαλύτερο.

Homework

Homework 2 - FizzBuzz

Αρχικά, γράψτε ένα πρόγραμμα σε **Python** το οποίο να τυπώνει τους αριθμούς από το **1** μέχρι το **100**.

Στη συνέχεια, τροποποιήστε το πρόγραμμα ώστε, για τους αριθμούς που είναι πολλαπλάσια του **3**, να τυπώνει τη λέξη "*Fizz*" (αντί για τον αντίστοιχο αριθμό) και για αριθμούς που είναι πολλαπλάσια του **5** τη λέξη "*Buzz*".

Αντί των αριθμών που είναι πολλαπλάσια και του **3** και του **5** (ταυτόχρονα) να τυπώνει "*FizzBuzz*".

Homework

Homework 3

Γράψτε μία συνάρτηση σε **Python**, η οποία να δέχεται μία φράση και να εμφανίζει μία μία τις λέξεις της φράσης αυτής.

```
>>> words('The Life of Brian')
The
Life
of
Brian
```


Χρήσιμα links

- 🔗 4. More Control Flow Tools — Python 3.10.4 documentation
<https://docs.python.org/3/tutorial/controlflow.html>
- 🔗 5. Data Structures — Python 3.10.4 documentation
<https://docs.python.org/3/tutorial/datastructures.html>

- 🔗 PEP 8 -- Style Guide for Python Code | Python.org
<https://www.python.org/dev/peps/pep-0008/>
- 🔗 How to Write Beautiful Python Code With PEP 8 – Real Python
<https://realpython.com/python-pep8/#tabs-vs-spaces>


Extra info

 Semicolon in Python - AskPython
<https://www.askpython.com/python/examples/semic...>

 python - Is there a built-in or more Pythonic way to try to parse a string to an integer - Stack Overflow
<https://stackoverflow.com/questions/2262333/is-there...>

 python - What is Truthy and Falsy? How is it different from True and False? - Stack Overflow
<https://stackoverflow.com/questions/39983695/what-i...>

 What is the difference between sort() and sorted()? - FAQ / Python FAQ - Codecademy Forums
<https://discuss.codecademy.com/t/what-is-the-differ...>

 Destructuring in Python
<https://blog.teclado.com/destructuring-in-python/>

THANK YOU!