

Εισαγωγή στην Επιστήμη Υπολογιστών

2η εργαστηριακή διάλεξη

Εισαγωγή στην γλώσσα Python

Επιμέλεια διαφανειών: Άννα Κεφάλα



Εισαγωγή στη γλώσσα Python – Β’ μέρος

Επαναλήψεις (while, for), Λίστες, Συναρτήσεις

- Λίστες
- Γιατί χρειαζόμαστε βρόχους (Loops): αυτοματοποίηση επαναλαμβανόμενων εργασιών
  - while: επανάληψη υπό συνθήκη (και πώς να αποφύγετε τον άπειρο βρόχο..)
  - for & range(): επανάληψη με γνωστό πλήθος βημάτων
- Συναρτήσεις (def): η αρχή DRY (Don't Repeat Yourself), επαναχρησιμοποιήσιμα modules, παράμετροι vs επιστρεφόμενες τιμές (return).

Λίστα (list): *σειρά από αριθμημένα κουτιά με δεδομένα (στη μνήμη)*



- Μία συλλογή δεδομένων, της οποίας κάθε θέση αποτελείται και από μία διαφορετική μεταβλητή/αντικείμενο
- Δυναμικό μέγεθος: δεν δηλώνουμε εκ των προτέρων πόσα στοιχεία χωράει, μεγαλώνει και μικραίνει ελεύθερα
- Ετερογενή στοιχεία: μπορεί να περιέχει ταυτόχρονα αριθμούς, λέξεις (strings), ακόμη και άλλες λίστες
- Δήλωση λίστας: `mylist = []` (*κενή*) ή `myinfo = [“Anna K.”, “Patission 76 st.”, 2108203000]` *κάθε στοιχείο χωρίζεται από το επόμενο με:* `,`
- Η αρίθμηση των θέσεων μίας λίστας ξεκινάει από το **0**



Διαχείριση Λίστας

- 1ο στοιχείο: `list[0]`, τελευταίο στοιχείο: `list[-1]`

```
fruits = ["μήλο", "μπανάνα", "πορτοκάλι"]
print(fruits[0])   # "μήλο"
print(fruits[-1])  # "πορτοκάλι“
print(fruits[3]) → τί θα εμφανίσει;;
```

- Μήκος λίστας (πλήθος στοιχείων): `len(όνομα λίστας)`

```
print(len(fruits))  # 3
```

- Προσθήκη στοιχείου στο τέλος της λίστας: `.append(στοιχείο)`

```
fruits.append("αχλάδι")
# fruits -> ["μήλο", "μπανάνα",
"πορτοκάλι", "αχλάδι"]
```

- Έλεγχος αν υπάρχει ένα στοιχείο στη λίστα: `in/not in`

```
if "μήλο" in fruits:
    print("Υπάρχει μήλο στη λίστα!")
```

Βασικές Μέθοδοι σε Λίστα

Μέθοδος	Περιγραφή
list.append(a)	Προσθέτει στο τέλος της λίστας το αντικείμενο a
list.remove(a)	Αφαιρεί από την λίστα το αντικείμενο a
del mylist[i]	Αφαιρεί από την λίστα με το όνομα mylist το στοιχείο της θέσης i
list.count(a)	Επιστρέφει τον αριθμό των φορών που εμφανίζεται το αντικείμενο a στην λίστα
list.clear()	Διαγράφει όλα τα στοιχεία της λίστας
list.insert(index,a)	Προσθέτει στη θέση index της λίστας το αντικείμενο a

Επαναληπτική δομή ελέγχου while



- **Τρόπος σύνταξης**:

```
while (expression):
    statements(s)
```

- Όσο η expression (συνθήκη ελέγχου) έχει τιμή **True**, τότε εκτελείται το block statements(s), σε εσοχή 4 χαρακτήρων. *Μπορεί να μην ξέρουμε εκ των προτέρων πόσες φορές θα επαναληφθεί το block, π.χ. ζητάμε από το χρήστη να τερματίσει την επανάληψη.*
- Αυτοματοποίηση επαναλαμβανόμενων tasks.
- Προσοχή στα **ατέρμονα loops** (βρόχοι)! Η συνθήκη πρέπει να γίνεται **False** για να τερματίζει το loop.

Επαναληπτική δομή ελέγχου while | παράδειγμα 1

- Ζητάω από το χρήστη να μου δώσει έναν αριθμό για να μετρήσω από το 0 μέχρι αυτό τον αριθμό.

```
count = 0
end_count = int(input("Μέχρι ποιο αριθμό θέλεις να μετρήσω; "))
while (count < end_count):
    print(f"Ο μετρητής έχει τιμή: {count}")
    count += 1
print(f"Τέλος! Μέτρησα μέχρι το {count}!")
```

- Τί θα κάνει ο κώδικας όταν τον τρέξω;

```
Μέχρι ποιο αριθμό θέλεις να μετρήσω; 10
Ο μετρητής έχει τιμή: 0
Ο μετρητής έχει τιμή: 1
Ο μετρητής έχει τιμή: 2
Ο μετρητής έχει τιμή: 3
Ο μετρητής έχει τιμή: 4
Ο μετρητής έχει τιμή: 5
Ο μετρητής έχει τιμή: 6
Ο μετρητής έχει τιμή: 7
Ο μετρητής έχει τιμή: 8
Ο μετρητής έχει τιμή: 9
Τέλος! Μέτρησα μέχρι το 10!
```

Επαναληπτική δομή ελέγχου while | παράδειγμα 2



- Ο κώδικας τυπώνει τα στοιχεία μίας λίστας, καθώς και την αντίστοιχη θέση που καταλαμβάνουν στην λίστα.

```
i=0
mylist = [1,3,5,7,9,11]
while (i < len(mylist)):
    print(f"Index of list: {i}, contains element: {mylist[i]}")
    i=i+1
print("end of list")
```

- Τί θα κάνει ο κώδικας όταν τον τρέξω;

```
Index of list: 0, contains element: 1
Index of list: 1, contains element: 3
Index of list: 2, contains element: 5
Index of list: 3, contains element: 7
Index of list: 4, contains element: 9
Index of list: 5, contains element: 11
end of list
```



Επαναληπτική δομή ελέγχου for

- Παρέχει τη δυνατότητα να διατρέχουμε επαναληπτικά μία ακολουθία (π.χ. μία List ή ένα String).
- **Σύνταξη**:

```
for iterating_var in sequence:
    statements(s)
```

- **Ερμηνεία**: Η τιμή κάθε αντικειμένου της sequence (η οποία έστω ότι είναι τύπου λίστας), ανατίθεται στην μεταβλητή iterating_var και στη συνέχεια εκτελείται το block statements(s).
- Η διαδικασία σταματά όταν εξαντληθούν τα αντικείμενα της λίστας.
- Iterating_var δεν χρειάζεται να δηλωθεί πριν.

Τελεστής **in/not in**

- Έλεγχος ύπαρξης τιμής σε ακολουθία (string, list, κ.α.) → επιστρέφει True ή False
- Σε δομή επανάληψης (for) για διάσχιση μιας ακολουθίας (π.χ. List)

Επαναληπτική δομή ελέγχου for | παραδείγματα



- Εμφάνιση των χαρακτήρων ενός string

```
string1 = "Python"
for char in string1:
    print(char)
```

Τί θα κάνει ο κώδικας όταν τον τρέξω;

```
P
y
t
h
o
n
```

- Εμφάνιση των στοιχείων μίας λίστας

```
students = ["Γιάννης", "Μαρία", "Νίκος"]
for s in students:
    print(f"Φοιτητής/τρια: {s}")
```

Τί θα κάνει ο κώδικας όταν τον τρέξω;

```
Φοιτητής/τρια: Γιάννης
Φοιτητής/τρια: Μαρία
Φοιτητής/τρια: Νίκος
```

Επαναληπτική δομή ελέγχου for | range() [1]

```
for iter_var in range(parameters):
    print(iter_var)
```

- Επανάληψη συγκεκριμένο πλήθος φορών ή παραγωγής μιας αριθμητικής ακολουθίας. Όλα τα ορίσματα της range() πρέπει να είναι τύπου int.

```
# range(stop), μόνο όριο τερματισμού,
ξεκινάει από το [0], τερματίζει stop-1
for i in range(5):
    print(f"με όριο τερματισμού 5: {i}")
```

```
με όριο τερματισμού 5: 0
με όριο τερματισμού 5: 1
με όριο τερματισμού 5: 2
με όριο τερματισμού 5: 3
με όριο τερματισμού 5: 4
```

```
# range(start, stop), ξεκινάει από start και
τερματίζει stop-1
for i in range(2, 6):
    print(i)
```

```
2
3
4
5
```

```
# range(start, stop, step), ξεκινάει από start,
βήμα αύξησης/μείωσης step και τερματίζει stop-1
for i in range(1, 10, 2):
    print(i)
```

```
1
3
5
7
9
```

Επαναληπτική δομή ελέγχου for | range() [2]

- Για να διατρέξουμε μια λίστα χρειαζόμαστε τη θέση (δείκτη / index) κάθε στοιχείου, συνδυάζουμε τη **range()** με τη **len()**

```
# διατρέχω μία λίστα, δίνοντας ως stop το μήκος της len(list)
students = ["Γιάννης", "Μαρία", "Νίκος"]

for i in range(len(students)):
    print(f"Θέση {i}: {students[i]}")
```



```
Θέση 0: Γιάννης
Θέση 1: Μαρία
Θέση 2: Νίκος
```

Συναρτήσεις (functions)

- Οι συναρτήσεις αποτελούν δομικό στοιχείο κάθε γλώσσας προγραμματισμού
- Ομαδοποιούν ένα σύνολο εντολών έτσι ώστε να μπορούν να χρησιμοποιηθούν πολλές φορές, με διαφορετική είσοδο, στο ίδιο πρόγραμμα.
- Μειώνουν το κόστος ανάπτυξης μιας εφαρμογής και βελτιώνουν το επίπεδο κατανόησης του κώδικα
- Μια συνάρτηση ορίζεται με την χρήση της δεσμευμένης λέξης **def**

```
# Ορισμός συνάρτησης χωρίς παραμέτρους
def greet():
    print("Καλώς ήρθατε στο εργαστήριο!")

# Κλήση της συνάρτησης
greet()
```

Εσοχή για το block εντολών της συνάρτησης

Συναρτήσεις | Παράδειγμα



- Συνάρτηση η οποία συγκρίνει 2 αριθμούς και εμφανίζει μήνυμα.

```
# Ορισμός με παραμέτρους
def compare_numbers(a, b):
    if b > a:
        print(f"Ο μεγαλύτερος αριθμός είναι το: {b}")
    elif a > b:
        print(f"Ο μεγαλύτερος αριθμός είναι το: {a}")
    else:
        print("Οι αριθμοί είναι ίσοι!")
# Κλήση με μεταβλητές
w = 3
s = 6
compare_numbers(w, s)
# Ή κλήση με απευθείας εκχώρηση τιμών
compare_numbers(12, 6)
```

Εκτέλεση κώδικα:

```
Ο μεγαλύτερος αριθμός είναι το: 6
Ο μεγαλύτερος αριθμός είναι το: 12
```

Συναρτήσεις | εντολή return



- Εντολή **return** επιστρέφει ένα αντικείμενο στο «κυρίως» πρόγραμμα, τερματίζεται η εκτέλεση της συνάρτησης

```
def find_max(a, b):
    if a > b:
        return a
    else:
        return b
# Αποθήκευση της επιστρεφόμενης τιμής σε μεταβλητή
result = find_max(12, 6)
print(f"Το αποτέλεσμα είναι: {result}")
```

Εκτέλεση κώδικα:

```
Το αποτέλεσμα είναι: 12
```

Συναρτήσεις | προκαθορισμένες τιμές παραμέτρων



- Μπορούμε να δώσουμε προκαθορισμένες τιμές στις παραμέτρους. Αν ο χρήστης δεν περάσει τιμή κατά την κλήση, χρησιμοποιείται η default.

```
# Χρήση προκαθορισμένων τιμών στις παραμέτρους
def student_info(name, department="Πληροφορική", year=1):
    print(f"Φοιτητής/τρια: {name} | Τμήμα: {department} | Έτος: {year}")

# Κλήσεις με διαφορετικούς τρόπους:
student_info("Μαρία")                    # Χρησιμοποιεί τα defaults για τμήμα και έτος
student_info("Νίκος", year=2)            # Χρήση ονομασίας ορίσματος (keyword argument)
student_info("Γιώργος", "Οικονομικά", 3)       # Παράκαμψη όλων των defaults
```

Εκτέλεση κώδικα:

```
Φοιτητής/τρια: Μαρία | Τμήμα: Πληροφορική | Έτος: 1
Φοιτητής/τρια: Νίκος | Τμήμα: Πληροφορική | Έτος: 2
Φοιτητής/τρια: Γιώργος | Τμήμα: Οικονομικά | Έτος: 3
```

Συναρτήσεις | εμβέλεια μεταβλητών



- **Τοπικές μεταβλητές** (local): μεταβλητές μέσα σε μια συνάρτηση υπάρχουν μόνο κατά την εκτέλεσή της και είναι ανεξάρτητες από μεταβλητές με το ίδιο όνομα εκτός αυτής

```
# Παράδειγμα εμβέλειας μεταβλητών, local vs global μεταβλητής
x = 50
def test_local(x):
    x = 2  # Αλλάζει μόνο το τοπικό x
    print("Τοπικό x:", x)
test_local(x)
print("Καθολικό x:", x)  # Παραμένει 50
```

Εκτέλεση κώδικα:

```
Τοπικό x: 2
Καθολικό x: 50
```

- **Καθολικές μεταβλητές** (global): αν θέλουμε μια συνάρτηση να τροποποιήσει μια μεταβλητή που ορίστηκε έξω από αυτήν, πρέπει να δηλωθεί ρητά με τη λέξη **global**

```
# Δήλωση της μεταβλητής counter ως global μέσα στη συνάρτηση
counter = 0
def increment():
    global counter
    counter = counter + 1
increment()
print(f"Η τιμή του counter είναι: {counter}")  # Τυπώνει 1
```

Εκτέλεση κώδικα:

```
Η τιμή του counter είναι: 1
```



Πρακτική εξάσκηση

ToDo στο eClass