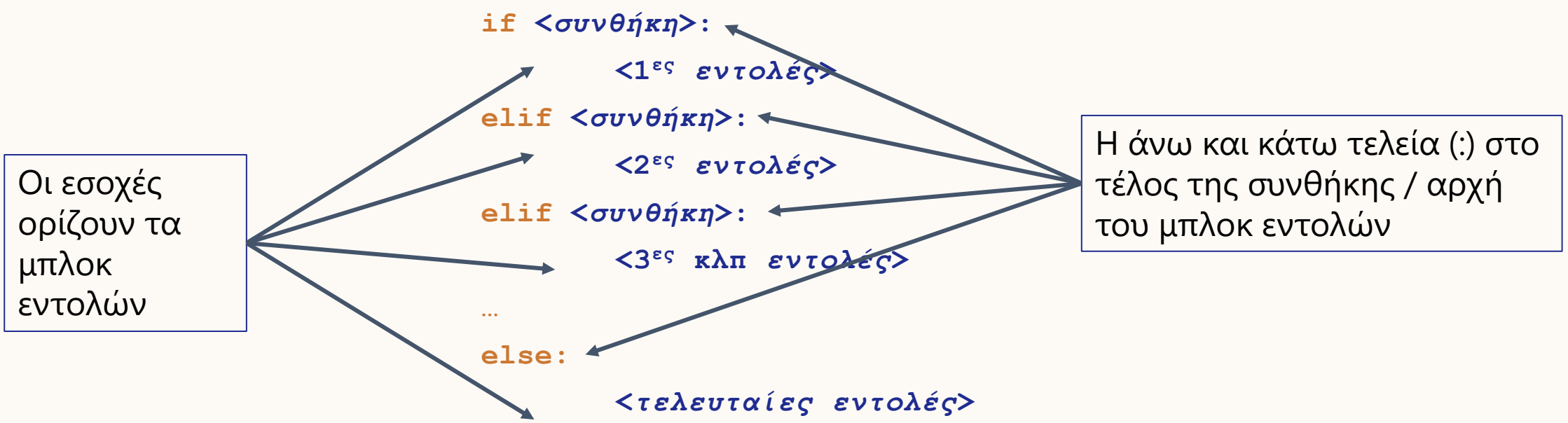
ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ Η/Υ
σε Python

Διάλεξη 3: Δομές επανάληψης

Μάρκος Ζάμπογλου, mzampoglou@aegean.gr

Υπενθύμιση: `if-elif-else`



Υπενθύμιση: False-like και True-like

False-like

- `False`
- `0`
- `None`
- `[], "", {}, {}`

True-like

- *Οτιδήποτε άλλο*

Υπενθύμιση: Τρεις τύποι σφαλμάτων

- Συντακτικά σφάλματα (Syntax errors)
- Λογικά σφάλματα (Logical errors)
- Σφάλματα χρόνου εκτέλεσης (Runtime errors)
  - Στην Python τα αποκαλούμε **Exceptions**

Υπενθύμιση: Διαχείριση σφαλμάτων

- **Syntax errors**
  - Τα εντοπίζει ο interpreter και προτείνει διορθώσεις
- **Logical errors**
  - Testing-testing-testing
- **Exceptions**
  - Testing και Exception handling

Υπενθύμιση: try…except…finally

```
a = input("Δώσε παρονομαστή: ")
try:
  denom = float(a)
  fraction = 1/denom
except ZeroDivisionError as e:
  print("Ο παρονομαστής δεν πρέπει να είναι 0.")
  fraction = None
except ValueError as e:
  print("Παρακαλώ δώστε αριθμό, όχι γράμματα.")
  fraction = None
except Exception as e:
  fraction = None
  print("Κάτι πήγε λάθος. Τύπος σφάλματος:", type(e), "Μήνυμα:", e)
finally:
  print("Το αποτέλεσμα είναι",fraction)
```

Δομές επανάληψης

`for` και `while`

Γιατί δομές επανάληψης;

Να γραφεί πρόγραμμα που θα ζητάει το βαθμό δέκα φοιτητών και συγκρίνει τον καθένα με το 5. Για κάθε φοιτητή, αν ο βαθμός είναι μεγαλύτερος ή ίσος του 5 να εμφανίζει «ΠΕΡΑΣΕ» αλλιώς «ΚΟΠΗΚΕ»

- Θα μπορούσαμε να γράψουμε δέκα φορές διαδοχικά τον ίδιο κώδικα, όμως αυτό θα ήταν άκομψο.
- Επίσης, τι θα γίνει αν στο μέλλον το μέγεθος της τάξης μεγαλώσει ή μικρύνει;
- Ή, αν αλλάξει η βάση από 5 σε 6;
- Θα έπρεπε να τροποποιούμε πολλές φορές το ίδιο πράγμα –πολύ πιθανό να γίνουν λάθη…

- Οι υπολογιστές είναι πολύ καλοί στο να επαναλαμβάνουν την ίδια διαδικασία πολλές φορές: **Δομές Επανάληψης**

Δομές Επανάληψης ή Βρόχοι (Loops)

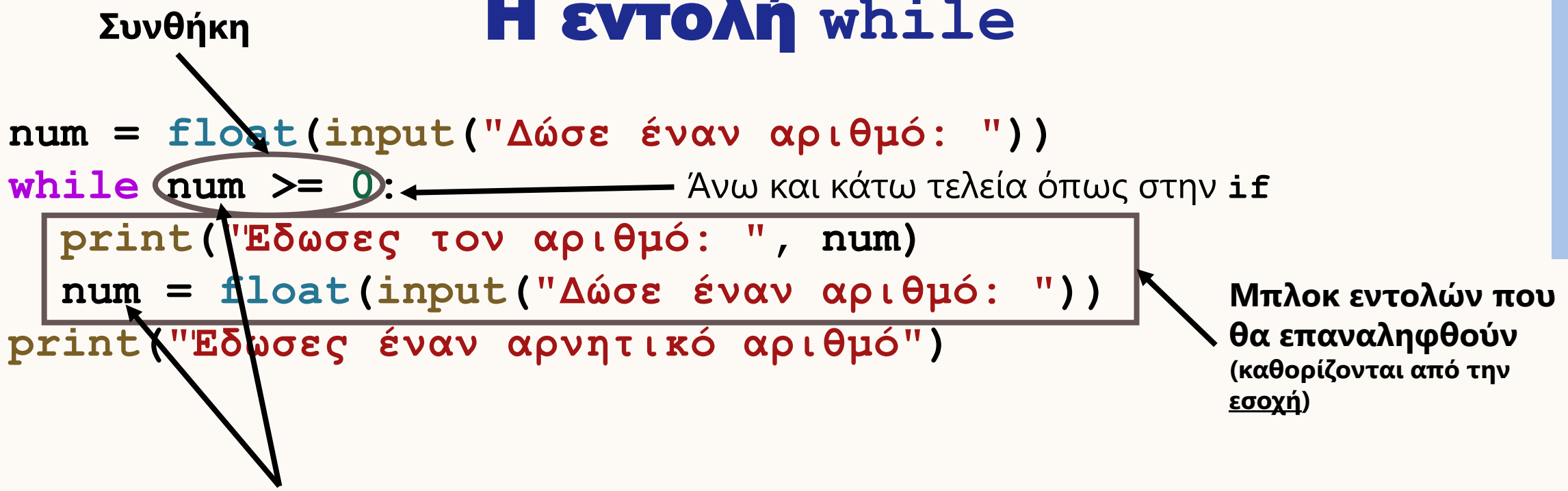
- Οι δομές επανάληψης της Python περιλαμβάνουν:
  - Ένα μπλοκ εντολών, αυτό που θέλουμε να επαναληφθεί (μπορεί να είναι και μόνο μία εντολή)
  - Μια εντολή στην αρχή του μπλοκ, η οποία πριν από κάθε επανάληψη ελέγχει αν θα ξανατρέξουμε τις εντολές του μπλοκ, ή αν η ροή εκτέλεσης θα πηδήξει στο τέλος τους

Δομές επανάληψης

Υπάρχουν δύο δομές επανάληψης στην Python:

1. Η δομή **`while`**
   - Συνεχίζει να επαναλαμβάνει ένα κομμάτι κώδικα όσο κάποια **συνθήκη** παραμένει **`True`**
2. Η δομή **`for`**
   - Επαναλαμβάνει ένα κομμάτι κώδικα για ένα πλήθος **διαφορετικών τιμών** μιας **μεταβλητής δείκτη**

Η εντολή while



Δομή `while`

- Η συνθήκη είναι μια έκφραση που επιστρέφει **`True`** ή **`False`**
  - Ή **`True`-like** ή **`False`-like**
  - Αν η συνθήκη είναι **`True`**, τότε η ακολουθία εντολών εκτελείται
  - Αν η συνθήκη είναι **`False`**, τότε η ροή των εντολών πηδάει στην εντολή μετά το τέλος του μπλοκ (δλδ της εσοχής)
- Αν φτάσουμε στο τέλος του μπλοκ, η ροή εντολών πηδάει πίσω στη συνθηκη
  - Οπότε και ξαναγίνεται ο έλεγχος
- Προσοχή: Αν την πρώτη φορά η συνθήκη είναι **`False`**, οι εντολές δε θα εκτελεστούν *καμία φορά*

Τι θα εμφανίσει ο κώδικας;

```
a = 5
while a>=0:
  print(a)
  a-=1
```

```
a = 5
while a>=0:
  print(a)
  a+=1
```

```
a = 5
b = 5
while a>=0:
  print(b)
  a-=1
```

```
a = 5
while a>=0:
  a-=1
  print(a)
```

```
a = 5
while a<=0:
  a-=1
  print(a)
```

```
a = 5
b = 5
while a>=0:
  print(b)
  b-=1
```

Η δομή `while...else`

- Η δομή της **`while`** μοιάζει με αυτήν της **`if`**
  - Η διαφορά είναι ότι στο τέλος του μπλοκ, επιστρέφουμε στον έλεγχο της **`while`** αντί να συνεχίζουμε προς τα κάτω.
- Παρόμοια με την **`if`**, η **`while`** μπορεί να ακολουθείται με μια εντολή **`else`**
  - Το μπλοκ της else εκτελείται μια φορά, όταν επιστρέψει **`False`** η συνθήκη

Παράδειγμα while...else

```
sum=0
a=1
while a>0:
    a = float(input("Δώσε θετικό αριθμό:"))
    sum+=a
else:
    print("Έδωσες αρνητικό ή μηδέν")
    print("Το άθροισμα ήταν:", sum)
```

Η εντολή `for`

- Η εντολή **`for`** προορίζεται να επιτελεί έναν συγκεκριμένο αριθμό επαναλήψεων, κατά τις οποίες μια μεταβλητή-δείκτης παίρνει διαδοχικές τιμές από ένα σύνολο που έχουμε ορίσει
- Χρησιμοποιεί τρία συστατικά
  1. Μια **μεταβλητή-δείκτη** (που συχνά την ονομάζουμε **`i`** από το *index*)
  2. Τον τελεστή **`in`**
  3. Ένα **αντικείμενο σύνθετου τύπου** (π.χ. μια λίστα)

```
for i in [1, 3, 5, 7]:
    ...
```

Η `for` στην Pyton vs άλλες γλώσσες

- Στις περισσότερες γλώσσες προγραμματισμού, η **`for`** δίνει τιμές σε μια μεταβλητή **από** μια τιμή, **έως** μια τιμή, με κάποιο **βήμα**
- Στην Python η **`for`** δίνει τιμές σε μια μεταβλητή από τα περιεχόμενα ενός σύνθετου αντικειμένου

```
for value in values:
    ...
```

Το **`values`** μπορεί να είναι λίστα, range, string, σύνολο, λεξικό, πλειάδα, κλπ

Σύνθετοι (*composite*) τύποι δεδομένων

- Ως τώρα έχουμε δουλέψει με τύπους δεδομένων που περιέχουν μόνο μια τιμή (μεταβλητή). Τέτοιοι είναι οι **`int`**, **`bool`**, **`float`** κλπ.
- Στον προγραμματισμό, υπάρχουν πολύ χρήσιμοι τύποι δεδομένων που αποτελούνται από πολλαπλές τιμές
- Στην Python, ένας τέτοιος είναι η **λίστα** (**`list`**)

```
#Μια λίστα που περιέχει τις τιμές 1, 6, 12, -2
a = [1, 6, 12, -2]
```

Θα δούμε τέτοιους τύπους σε βάθος σε επόμενες διαλέξεις

Ο τελεστής `in` ως λογική συνθήκη

Ο τελεστής **`in`** έχει δύο διαφορετικές χρήσεις στην Python

1. Στην πρώτη περίπτωση **επιστρέφει μια λογική τιμή**:

   Η έκφραση **`x in Y`**, όπου **`x`** μια τιμή και **`Y`** ένας σύνθετος τύπος δεδομένων, ισούται με **`True`** αν η τιμή **`x`** βρίσκεται μέσα στο **`Y`** και **`False`** αν όχι

Π.χ. η έκφραση **`4 in [1, 6, 4, 12]`** επιστρέφει **`True`**

Βλ. κώδικα

Ο τελεστής `in` στις δομές `for`

2. Η δεύτερη αφορά **στη δομή επανάληψης `for`**

   Ως τμήμα μιας εντολής **`for`**, η in επιστρέφει μία-μία τις τιμές του σύνθετου αντικειμένου, διαδοχικά σε κάθε επανάληψη και τις αναθέτει στη μεταβλητής-δείκτη

Πχ. η εντολή `for i in [1, 3, 5, 7]:` θα εκτελέσει το μπλοκ κώδικα **4** φορές, με το **`i`** να παίρνει διαδοχικά τις τιμές **1**, **3**, **5**, **7**.

Βλ. κώδικα

Η εντολή `for`

```
for i in [1, 3, 5, 7]:
    print(i+1)
```

- Συχνά θέλουμε η **μεταβλητή-δείκτης** να πάρει τιμές από μια πολύ μεγάλη εμβέλεια, π.χ. από **1** έως **10.000**
  - Δε γίνεται να τις γράψουμε όλες σε μια λίστα
- Λύση: συνάρτηση **`range()`**

Η συνάρτηση `range()`

Η **`range()`** επιστρέφει ένα αντικείμενο τύπου **`range`** που περιέχει μια ακολουθία τιμών:

1. Αν την καλέσω με μία παράμετρο ως **`range(stop)`**, όπου **`stop`** ένας ακέραιος, το αντικείμενο θα περιέχει τις τιμές από **`0`** έως **`stop`** (χωρίς να συμπεριλαμβάνει τη **`stop`**)
2. Αν την καλέσω με δύο παραμέτρους ως **`range(start, stop)`**, θα περιέχει τις τιμές από **`start`** έως **`stop`** (χωρίς να συμπεριλαμβάνει τη **`stop`**)
3. Αν την καλέσω με τρεις παραμέτρους ως **`range(start, stop, step)`**, θα περιέχει τις τιμές από **`start`** έως **`stop`** (χωρίς να συμπεριλαμβάνει τη **`stop`**) με βήμα **`step`**.

```
range(5): 0,1,2,3,4
range(5,9): 5,6,7,8
range(6,2,-1): 6,5,4,3
```

Απεικονίζοντας τις τιμές μιας `range()`

- Παρότι δεν έχουμε δει ακόμα τις λίστες σε βάθος, είπαμε ήδη τα βασικά
- Καθώς ξέρουμε ότι είναι ένας τύπος αντικειμένου, σημαίνει ότι μπορούμε να μετατρέψουμε αντικείμενα άλλου τύπου σε λίστες με τη συνώνυμη συνάρτηση **`list()`**
  - Όπως η **`int("6")`** επιστρέφει **`6`**
  - Έτσι και η **`list(range(3,-15,-2))`** επιστρέφει `[3, 1, -1, -3, -5, -7, -9, -11, -13]`

Τι θα εμφανίσει ο κώδικας;

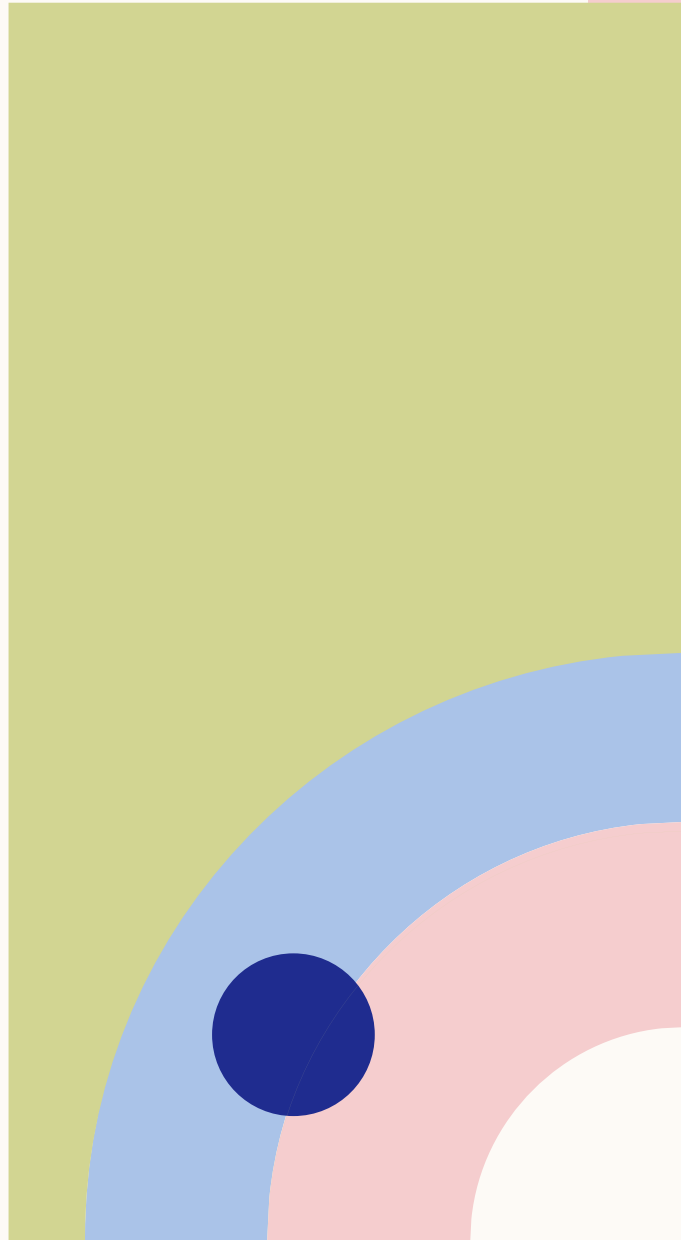
```
print(list(range(4)))

print(list(range(1,4)))

print(list(range(1,14,3)))

print(list(range(1,-14,-3)))

print(list(range(1,-14,3)))
```



`while` και `for`

- Η **`while`** είναι μια επανάληψη που βασίζεται σε έναν λογικό έλεγχο
  - Βέβαια, και η **`for`**, αν το σκεφτούμε, βασίζεται στον έλεγχο αν υπάρχουν ακόμα στοιχεία στο σύνθετο αντικείμενο από το οποίο αντλεί τιμές
- Η βασική τους διαφορά είναι ότι η **`for`** έχει την τιμή-δείκτη, και αντλεί τιμές μέχρι να αδειάσει τη δομή δεδομένων
  - Ενώ η συνθήκη τερματισμού της **`while`** είναι πιο ευέλικτη
- **ΠΡΟΣΟΧΗ**: Σε άλλες γλώσσες προγραμματισμού, λέμε ότι η **`for`** προορίζεται να εκτελέσει έναν «πεπερασμένο αριθμό επαναλήψεων»
  - Στην Python αυτό δεν ισχύει: *μπορούμε να προσθέτουμε νέα αντικείμενα στη λίστα κατά τη διάρκεια των επαναλήψεων*!
    - Θα τα δούμε όταν μιλήσουμε για λίστες

Τι θα εμφανίσει ο κώδικας;

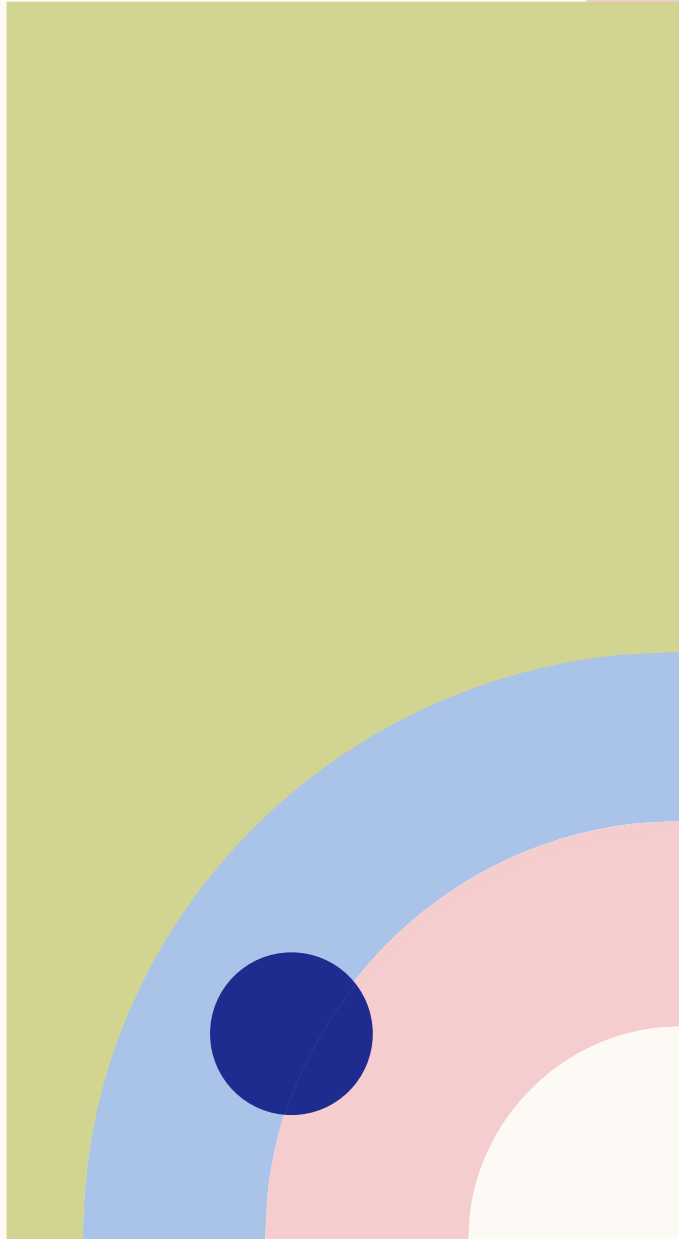
```
for a in range(4):
  print(a*2)

for element in ["Η", "Python", "είναι", "φοβερή"]:
  print(element)

for a in []:
  print(a)

for el in [5, 21, "Μάρκος", 2]:
  print(el + 2)

a=0
for b in [2, 4, 6, 8]:
  print(a+b)
  a = b
```



Μετρητές, αθροιστές, γινόμενα (1)

- Στον προγραμματισμό, συχνά χρειάζεται να μετράμε, να αθροίζουμε, ή να πολλαπλασιάζουμε στοιχεία
- Παρότι η Python παρέχει ισχυρά εργαλεία για αυτές τις λειτουργίες, καλό είναι να έχουμε μια βασική εικόνα πώς μπορούμε να το πετύχουμε και με δικές μας εντολές

Μετρητές, αθροιστές, γινόμενα (2)

- **Μέτρηση**
  - Αρχικοποιούμε μια μεταβλητή-**μετρητή** (π.χ. **`count`**) σε 0
  - Κάθε φορά που συναντάμε αυτό που θέλουμε να μετρήσουμε, αυξάνουμε το μετρητή κατά **`1`**
    `count+=1`
- **Άθροισμα**
  - Αρχικοποιούμε έναν **αθροιστή** (π.χ. **`sum`**) σε 0
  - Κάθε φορά που παίρνουμε έναν αριθμό **`a`** που θέλουμε να αθροίσουμε, αυξάνουμε τον αθροιστή κατά **`a`**
    `sum+=a`
- **Γινόμενο**
  - Αρχικοποιούμε το **γινόμενο** (π.χ. **`prod`**) στο `1`
  - Κάθε φορά που παίρνουμε έναν νέο αριθμό **`a`**, τον πολλαπλασιάζουμε στο γινόμενο
    `prod*=a`

Παράδειγμα 3.1

Να γραφεί κώδικας σε Python που να ζητάει από το χρήστη έναν αριθμό a, και στη συνέχεια να του ζητάει a αριθμούς, και να επιστρέφει το άθροισμά τους και το μέσο όρο τους.

Παράδειγμα 3.1 – Λύση με `for`

Να γραφεί κώδικας σε Python που να ζητάει από το χρήστη έναν αριθμό a, και στη συνέχεια να του ζητάει a αριθμούς, και να επιστρέφει το άθροισμά τους και το μέσο όρο τους.

```
sum = 0
a = int(input("Πόσους αριθμούς να διαβάσω:"))
for i in range(a):
  b = float(input("Δώσε έναν αριθμό:"))
  sum += b

print(sum)
mo=sum/a
print(mo)
```

Παράδειγμα 3.1 – Λύση με `while`

Να γραφεί κώδικας σε Python που να ζητάει από το χρήστη έναν αριθμό a, και στη συνέχεια να του ζητάει a αριθμούς, και να επιστρέφει το άθροισμά τους και το μέσο όρο τους.

```
sum = 0
a = int(input("Πόσους αριθμούς να διαβάσω:"))
count = 0
while count<a:
  b = float(input("Δώσε έναν αριθμό:"))
  sum += b
  count += 1
print(sum)
mo=sum/a
print(mo)
```

Μέγιστο & ελάχιστο

- Αν μας ζητείται να βρούμε τον ελάχιστο από ένα πλήθος αριθμών, αρχικοποιούμε σε μια τιμή μεγαλύτερη από οποιαδήποτε πιθανή τιμή
- Για κάθε νέα τιμή που λαμβάνουμε, ελέγχουμε αν είναι μικρότερη από την έως τώρα μικρότερη. Αν ναι, **αντικαθιστούμε**
  - Αν αναζητάμε τη μεγαλύτερη, αντιστρέφουμε την αρχικοποίηση και τη σύγκριση
- **Τι γίνεται αν οι αριθμοί δεν έχουν άνω ή κάτω όριο;**
  - Στην Python έχω τη δυνατότητα να θέσω την τιμή ενός αριθμού στο **∞** ή **-∞**

Παράδειγμα 3.2

Να γραφεί κώδικας σε Python που να διαβάζει 10 θετικούς αριθμούς και να εμφανίζει το μεγαλύτερο από αυτούς.

Παράδειγμα 3.2 - Λύση

Να γραφεί κώδικας σε Python που να διαβάζει 10 θετικούς αριθμούς και να εμφανίζει το μεγαλύτερο από αυτούς.

```
# Άσκηση 3.2
max = 0
for i in range(10):
  a = float(input("Δώσε έναν αριθμό: "))
  if max<a:
    max = a
print(max)
```

Παράδειγμα 3.3

Να γραφεί κώδικας σε Python που να διαβάζει 10 οποιουσδήποτε αριθμούς και να εμφανίζει το μεγαλύτερο από αυτούς.

Παράδειγμα 3.3 – Λύση 1

Να γραφεί κώδικας σε Python που να διαβάζει 10 οποιουσδήποτε αριθμούς και να εμφανίζει το μεγαλύτερο από αυτούς.

```
max = -float("inf")
for i in range(10):
  a = float(input("Δώσε έναν αριθμό: "))
  if max<a:
    max = a
print(("Ο μεγαλύτερος ήταν ο:", max)
```

Ένας τρόπος να δώσω τιμή "άπειρο" (inf) σε μια μεταβλητή είναι να μετατρέψω σε **float** το **string** "inf"

Παράδειγμα 3.3 – Λύση 2

Να γραφεί κώδικας σε Python που να διαβάζει 10 οποιουσδήποτε αριθμούς και να εμφανίζει το μεγαλύτερο από αυτούς.

```
import math
max = -math.inf
for i in range(10):
  a = float(input("Δώσε έναν αριθμό: "))
  if max<a:
    max = a
print(("Ο μεγαλύτερος ήταν ο:", max)
```

Ένας άλλος τρόπος να δώσω τιμή "άπειρο" (inf) σε μια μεταβλητή είναι να εισαγάγω τη βιβλιοθήκη **math** που περιέχει τη σταθερά **math.inf**

inf και nan

```
import math
x = -math.inf
print(type(x))
<class 'float'>
b = x/x
print (b)
nan
print(type(b))
<class 'float'>
```

nan από τα αρχικά Not A Number, για τους αριθμούς που δεν ορίζονται

Τόσο το **nan** όσο και **inf** το είναι τύπου **float**

Ανάθεση και σύγκριση nan

```
import math
a = math.nan
b = math.nan
print(a==b)
```
```
False
```

(!)

```
import math
a = math.nan
print(math.isnan(a))
```
```
True
```

Η **nan** ποτέ δεν αποκρίνεται **True** σε ισότητα –*δε μπορεί να συγκριθεί γιατί δεν είναι αριθμός*.
Ο μόνος τρόπος να την ανιχνεύσουμε είναι με τη συνάρτηση **isnan()** της βιβλιοθήκης **math**.

Σύγκριση inf

```
import math
x = -math.inf
b = math.inf
print(b==x)
False
print(b==-x)
True
```

```
import math
x = -math.inf
print(math.isinf(x))
True
print(math.isinf(-x))
True
```

Η **math.isinf(a)** επιστρέφει **True** είτε το **a** είναι **inf** είτε είναι **-inf**

`break` και `continue`

- Η εντολή **`break`** τερματίζει ολόκληρη τη δομή επανάληψης και στέλνει τη ροή εκτέλεσης έξω από αυτήν
- Η εντολή **`continue`** τερματίζει την τρέχουσα επανάληψη και στέλνει τη ροή εκτέλεσης στο επόμενο βήμα της

Παράδειγμα 3.4

Να γραφεί πρόγραμμα που θα δέχεται από το χρήστη 10 αριθμούς ή και λιγότερους, αν το άθροισμά τους ξεπεράσει το 100

Παράδειγμα 3.4 – Λύση με `while`

Να γραφεί πρόγραμμα που θα δέχεται από το χρήστη 10 αριθμούς ή και λιγότερους, αν το άθροισμά τους ξεπεράσει το 100

Λύση με **`while`** που ελέγχει και το άθροισμα και το πλήθος

```
i=0
sum=0
while i<=10 and sum<=100:
    num = float(input("Δώσε τον
    επόμενο αριθμό:"))
    sum += num
    i += 1
print("Τελικό άθροισμα:", sum)
```

Παράδειγμα 3.4 – Λύση με `for/break`

Να γραφεί πρόγραμμα που θα δέχεται από το χρήστη 10 αριθμούς ή και λιγότερους, αν το άθροισμά τους ξεπεράσει το 100

Λύση με **`for`** που διακόπτεται αν το άθροισμα ξεπεράσει το 100

```
sum=0
for i in range(10):
    num = float(input("Δώσε τον επόμενο αριθμό:"))
    sum += num
    if (sum>100):
        break
print("Τελικό άθροισμα:", sum)
```

Παράδειγμα 3.4 – Λύση με `while/break`

Να γραφεί πρόγραμμα που θα δέχεται από το χρήστη 10 αριθμούς ή και λιγότερους, αν το άθροισμά τους ξεπεράσει το 100

Λύση με **`while True`** που διακόπτεται για οποιαδήποτε από τις δυο συνθήκες

```
sum=0
i=0
while True:
    num = float(input("Δώσε τον επόμενο αριθμό:"))
    sum += num
    i+=1
    if (sum>100 or i==10):
        break
print("Τελικό άθροισμα:", sum)
```

Ατέρμονος βρόχος;

Όχι!

Παράδειγμα 3.5

Να γραφεί πρόγραμμα που θα δέχεται 10 ακέραιους αριθμούς από το χρήστη, και θα εντοπίζει τον μικρότερο από τους περιττούς, αγνοώντας τους άρτιους.

Παράδειγμα 3.5 – Λύση με `for` και `if`

Να γραφεί πρόγραμμα που θα δέχεται 10 ακέραιους αριθμούς από το χρήστη, και θα εντοπίζει τον μικρότερο από τους περιττούς, αγνοώντας τους άρτιους.

```
import math
min = math.inf
for i in range(10):
    num = int(input("Δώσε έναν
                        ακέραιο:"))
    if num%2==1:
        if num<min:
            min=num

print("Ο μικρότερος άρτιος
                        ήταν:", min)
```

Παράδειγμα 3.5 – Λύση με for και continue

Να γραφεί πρόγραμμα που θα δέχεται 10 ακέραιους αριθμούς από το χρήστη, και θα εντοπίζει τον μικρότερο από τους περιττούς, αγνοώντας τους άρτιους.

```
import math
min = math.inf
for i in range(10):
    num = int(input("Δώσε έναν
                        ακέραιο:"))
    if num%2==0:
        continue
    if num>=min:
        continue
    min=num
print("Ο μικρότερος άρτιος
                        ήταν:", min)
```

Too many options - ξανά

- Υπερβολικά πολλές επιλογές για να πετυχαίνουμε *το ίδιο πράγμα*;
- Εν μέρει ναι –σε αρχάριο επίπεδο, αυτές οι διαφορετικές λύσεις είναι πρακτικά ισοδύναμες
- Σε πιο προχωρημένο επίπεδο υπάρχουν διαφορές –π.χ. στο υπολογιστικό κόστος ή στην κομψότητα του κώδικα
- Για την ώρα ας μάθουμε να *καταλαβαίνουμε* όλες τις λύσεις –και να εφαρμόζουμε όποια προτιμούμε

Η εντολή pass

- Η εντολή pass δεν κάνει **τίποτα**
- Είναι χρήσιμη κατά τη διάρκεια της συγγραφής κώδικα, όταν θέλουμε π.χ. να δημιουργήσουμε μια δομή, αλλά δε ξέρουμε ακόμα τι θα κάνουμε σε κάθε σενάριο
  - Απαγορεύεται να έχουμε κενό μπλοκ

```
a = int(input("Δώσε έναν αριθμό:"))
if a%2==0:
  pass
else:
  print("Περιττός")
```

Σκόπιμο «κλείδωμα» του συστήματος

Να γραφεί κώδικας που θα ζητάει να δώσουμε τον κωδικό. Αν δώσουμε τρεις φορές εσφαλμένη απάντηση, να κλειδώνει το σύστημα.

```
password = "123456"
tries = 0
while tries<3:
    attempt=input("Δώσε κωδικό")
    if attempt==password:
        print("Καλωσήρθατε")
        break
    tries+=1
else:
    print("Κλείδωμα συσκευής")
    while True:
        pass
```

Αναζήτηση

- Μου ζητείται να απαντήσω αν μία τιμή μέσα σε ένα σύνολο τιμών πληροί κάποια προϋπόθεση
  - Αν ένας από τους αριθμούς που έδωσε ο χρήστης ισούται με 5
  - Αν ένας από τους αριθμούς που έδωσε ο χρήστης είναι μικρότερος από τον προηγούμενο
  - …
- Η κλασική προσέγγιση γίνεται με μια μεταβλητή **`bool`** που παίζει το ρόλο σημαίας (flag)

Παράδειγμα 3.6

Να γραφεί πρόγραμμα που θα δέχεται μέχρι και 10 ακέραιους αριθμούς από το χρήστη, αλλά θα σταματάει αμέσως αν δεχτεί το 0. Μετά το τέλος της εκτέλεσης, θα εμφανίζει ένα μήνυμα που θα εξηγεί αν έγιναν δέκα επαναλήψεις ή αν ο αλγόριθμος τερμάτισε νωρίτερα.

Παράδειγμα 3.6 – Λύση με flag

Να γραφεί πρόγραμμα που θα δέχεται μέχρι και 10 ακέραιους αριθμούς από το χρήστη, αλλά θα σταματάει αμέσως αν δεχτεί το 0. Μετά το τέλος της εκτέλεσης, θα εμφανίζει ένα μήνυμα που θα εξηγεί αν έγιναν δέκα επαναλήψεις ή αν ο αλγόριθμος τερμάτισε νωρίτερα.

Η `found` είναι η «σημαία». Ξεκινάει «κατεβασμένη».

```
i=0
found=False
while i<10 and found==False:
  x=int(input("Δώσε ακέραιο:"))
  if x==0:
    print("Έδωσες μηδέν")
    found = True
  i+=1
if not found:
  print("Έγιναν δέκα επαναλήψεις")
```

Παράδειγμα 3.6 – Λύση με `while..else`

Να γραφεί πρόγραμμα που θα δέχεται μέχρι και 10 ακέραιους αριθμούς από το χρήστη, αλλά θα σταματάει αμέσως αν δεχτεί το 0. Μετά το τέλος της εκτέλεσης, θα εμφανίζει ένα μήνυμα που θα εξηγεί αν έγιναν δέκα επαναλήψεις ή αν ο αλγόριθμος τερμάτισε νωρίτερα.

```
i=0
while i<10:
  x=int(input("Δώσε ακέραιο:"))
  if x==0:
    print("Έδωσες μηδέν")
    break
  i+=1
else:
  print("Έγιναν δέκα επαναλήψεις")
```

Παράδειγμα 3.7

Να γραφεί κώδικας που να ζητάει από το χρήστη έναν αριθμό και να απαντάει αν είναι πρώτος.

Πρώτοι λέγονται οι αριθμοί που διαιρούνται ακέραια μόνο με τον εαυτό τους και με τη μονάδα.

Παράδειγμα 3.7

Να γραφεί κώδικας που να ζητάει από το χρήστη έναν αριθμό και να απαντάει αν είναι πρώτος.

Πρώτοι λέγονται οι αριθμοί που διαιρούνται ακέραια μόνο με τον εαυτό τους και με τη μονάδα.

```
a = int(input("Δώσε
    έναν ακέραιο"))
i=2
while i<=a/2:
    if a%i==0:
        print("Δεν είναι
              πρώτος")
        break
    i+=1
else:
    print("Είναι πρώτος")
```

Εμφωλευμένη επανάληψη

Αφού μέσα στο μπλοκ μπορούμε να έχουμε ό,τι εντολές θέλουμε, μπορούμε να έχουμε και νέες δομές επανάληψης

```
#Προπαίδεια
for i in range(1,10):
  for j in range(1,10):
    print(i,'x', j, '=', i*j)
```

Παράδειγμα 3.8

Να γραφεί κώδικας
που θα δέχεται
έναν ακέραιο **a**, και
για κάθε ακέραιο
από το **2** ως το **a**
θα γράφει αν είναι
πρώτος ή όχι.

Παράδειγμα 3.8 – Λύση

Να γραφεί κώδικας που θα δέχεται έναν ακέραιο **a**, και για κάθε ακέραιο από το **2** ως το **a** θα γράφει αν είναι πρώτος ή όχι.

```
a = int(input("Δώσε έναν ακέραιο"))
for i in range(2,a+1):
  j=2
  while j<=i/2:
    if i%j==0:
      print("Ο αριθμός", i, "δεν
                          είναι πρώτος")
      break
    j+=1
  else:
    print("Ο αριθμός", i, "είναι
                          πρώτος")
```

Έλεγχος εγκυρότητας

- Πάρα πολλά προβλήματα απαιτούν συγκεκριμένο τύπο δεδομένων
  - «Να ζητάει από το χρήστη έναν ακέραιο»
- Δε μπορούμε να είμαστε βέβαιοι ότι ο χρήστης θα συμμορφωθεί
  - Μπορεί να δώσει ακόμα και κείμενο
- Μία λύση: Exception handling
- Δεύτερη λύση: κλείνουμε την **`input()`** σε ένα βρόχο ο οποίος διακόπτεται μόνο όταν η είσοδος είναι η αναμενόμενη

Παράδειγμα 3.9

Να γραφεί πρόγραμμα που θα δέχεται 10 θετικούς αριθμούς από το χρήστη, και θα εντοπίζει τον μεγαλύτερο

Να γίνεται έλεγχος εγκυρότητας σχετικά με το αν ο αριθμός είναι θετικός ή όχι.

Παράδειγμα 3.9 – Λύση

Να γραφεί πρόγραμμα που θα δέχεται 10 θετικούς αριθμούς από το χρήστη, και θα εντοπίζει τον μεγαλύτερο

Να γίνεται έλεγχος εγκυρότητας σχετικά με το αν ο αριθμός είναι θετικός ή όχι.

```
max = 0
for i in range(10):
    num = int(input("Δώσε έναν θετικό
                                ακέραιο:"))
    while num<=0:
        num = int(input("Είπαμε
                                θετικό:"))
    if num>max:
        max=num
print("Ο μεγαλύτερος αριθμός ήταν:",
                                    max)
```

Έλεγχος εγκυρότητας στον πραγματικό κόσμο

- Όπως έχουμε πει, η **`input()`** δεν είναι ο τρόπος που δεχόμαστε δεδομένα συνήθως στις πραγματικές εφαρμογές
- Η λογική του ελέγχου εγκυρότητας όμως επεκτείνεται πολύ πιο πέρα από την **`input()`**
- Π.χ. σε μια σύνδεση μέσω internet, πολύ συχνά ένα πακέτο πληροφορίας φτάνει αλλοιωμένο
  - Εφαρμόζουμε κι εκεί βρόχους που το ξαναζητάνε έως ότου να έρθει χωρίς σφάλματα

Παράδειγμα 3.10

Να γραφεί κώδικας που αρχικά θα αποθηκεύει στη μνήμη μια τυχαία τιμή από 1 έως 100 με τις εντολές:

```
import random
target = random.randint(1, 100)
```

Στη συνέχεια θα δίνει στον χρήστη 7 ευκαιρίες να μαντέψει τον αριθμό. Σε κάθε προσπάθεια του παίχτη, το σύστημα θα απαντάει «χαμηλότερα» ή «ψηλότερα» αναλόγως αν ο στόχος είναι χαμηλότερα ή ψηλότερα από τον αριθμό που δοκίμασε ο χρήστης.

Αν ο χρήστης μαντέψει σωστά τον αριθμό, ή αν ξοδέψει και τις 7 ευκαιρίες του, το παιχνίδι θα τελειώνει και θα εμφανίζει στο χρήστη το μήνυμα "κέρδισες" ή "έχασες", ανάλογα με το τι συνέβη.

Παράδειγμα 3.10 – Λύση

```
import random
target = random.randint(1, 100)
tries = 1
while tries<=7:
    guess = int(input("Προσπάθησε να μαντέψεις τον αριθμο:"))
    if guess<target:
        print("Δοκίμασε ψηλότερα.")
    elif guess>target:
        print("Δοκίμασε χαμηλότερα.")
    else:
        print("Το βρήκες με την",tries,"η προσπάθεια")
        break
    tries+=1
else:
    print("Λυπάμαι, κάηκες.")
```

Παράδειγμα 3.11

Να γραφεί κώδικας σε Python που θα δέχεται τρεις αριθμούς **a**, **r**, **n**, και θα τυπώνει τους **n** πρώτους όρους της γεωμετρικής προόδου με πρώτο όρο **a** και λόγο **r**.

$$a_n = ar^{n-1}$$

Παράδειγμα 3.11

Να γραφεί κώδικας σε Python που θα δέχεται τρεις αριθμούς **a**, **r**, **n**, και θα τυπώνει τους **n** πρώτους όρους της γεωμετρικής προόδου με πρώτο όρο **a** και λόγο **r**.

$$a_n = ar^{n-1}$$

```
a = float(input("Πρώτος όρος: "))
r = float(input("Λόγος: "))
n = int(input("Πλήθος όρων: "))
for i in range(1,n+1):
  print(a*r**(i-1))
```