ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ Η/Υ
σε Python

Διάλεξη 2: Δομές Επιλογής

Μάρκος Ζάμπογλου, mzampoglou@aegean.gr

Υπενθύμιση: Δομικά στοιχεία της Python

- Μεταβλητές/Αντικείμενα
- Τελεστές
- Συναρτήσεις
- Δηλώσεις
- Εκφράσεις

Δεσμευμένες λέξεις της Python

and	else	in	return
as	except	is	True
assert	False	lambda	try
break	finally	None	while
class	for	nonlocal	with
continue	from	not	yield
def	global	or	
del	if	pass	
elif	import	raise	

Τύποι δεδομένων στην Python

- **Numeric (Αριθμητικός)**
  - **Integer (Ακέραιος):** 0, 4, 1819, -40
  - **Float (Πραγματικός):** 6.3, 5.0, 519.32, -11.61
  - **Complex (Μιγαδικός):** 2+4j, 3.2-2.1j
- **Boolean (Λογικός):** True, False
- **String (Συμβολοσειρά):** "Hello World", "Καλημέρα", "4+4=8"

Μία τιμή, "*βαθμωτές*" ("*scalar*") μεταβλητές

- **List (Λίστα):** [2, 3.14, 'b', [33, 49], False]
- **Set (Σύνολο):** {2,3,4, 'Γιώργος', 3.14}
- **Tuple (Πλειάδα):** (3, 4, 'c')
- **Dictionary (Λεξικό):** {'Name': 'Markos', 'Height': 1.94}

Πολλές τιμές, "*σύνθετα*" ("*composite*") αντικείμενα από διάφορους άλλους τύπους

Υπενθύμιση: τελεστές πράξεων

- **Αριθμητικοί**: **+**, **-** , ***** , **/**, **//**, **%**, ******
- **Συγκριτικοί: ==** , **<** , **>** , **<=** , **>=** , **!=**
- **Λογικοί:**
  - **and:** επιστρέφει **`True`** μόνο αν και οι δύο τιμές είναι **`True`**
  - **or:** επιστρέφει **`True`** αν έστω και μία από τις δύο τιμές είναι **`True`**
  - **not:** δέχεται μία τιμή και επιστρέφει την αντίστροφή της

Ενσωματωμένες συναρτήσεις (Built-in Functions)

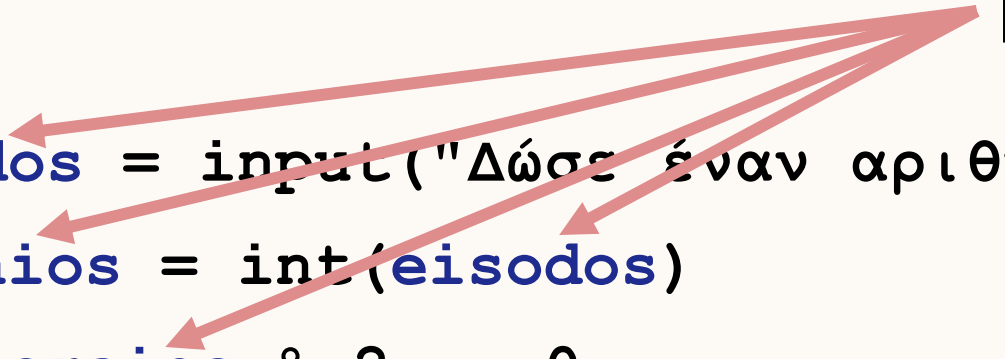
- `print(x)`
- `input(x)`
- `max(x,y,z,a…)`
- `min(x,y,z,a…)`
- `pow(x,y)`
- `int(x)`
- `abs(x)`
- `round(x)`
- …

Ένα πρόγραμμα σε Python

```
eisodos = input("Δώσε έναν αριθμό: ")
akeraios = int(eisodos)
if akeraios % 2 == 0:
  print("Ο αριθμός είναι άρτιος")
else:
  print("Ο αριθμός είναι περιττός")
```

Ένα πρόγραμμα σε Python

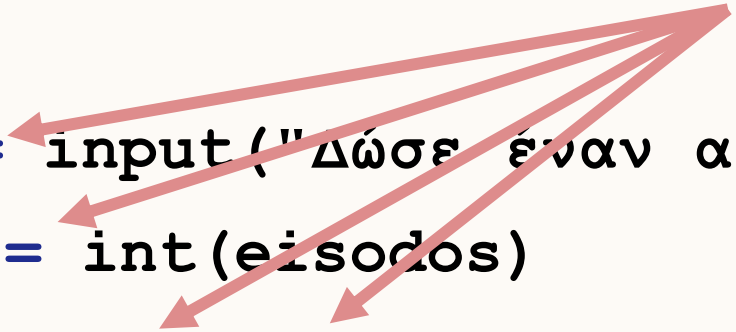
Μεταβλητές



```
eisodos = input("Δώσε έναν αριθμό: ")
akeraios = int(eisodos)
if akeraios % 2 == 0:
  print("Ο αριθμός είναι άρτιος")
else:
  print("Ο αριθμός είναι περιττός")
```

Ένα πρόγραμμα σε Python

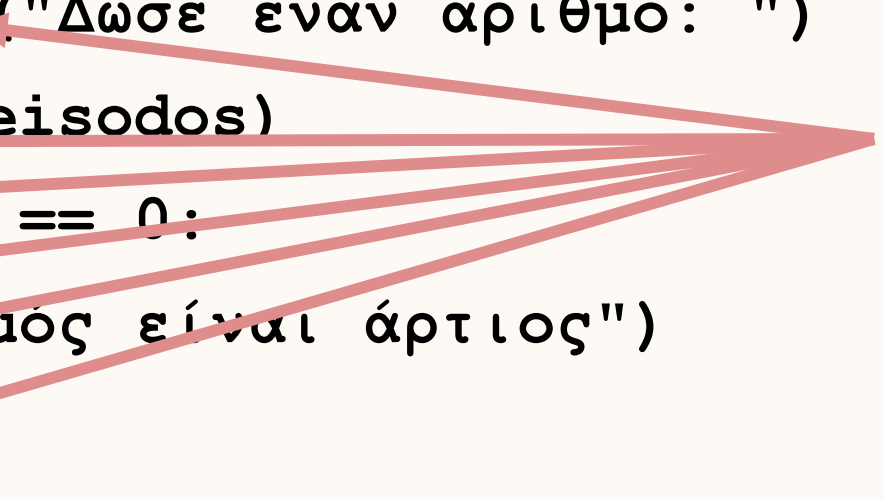
Τελεστές



```
eisodos = input("Δώσε έναν αριθμό: ")
akeraios = int(eisodos)
if akeraios % 2 == 0:
  print("Ο αριθμός είναι άρτιος")
else:
  print("Ο αριθμός είναι περιττός")
```

Ένα πρόγραμμα σε Python

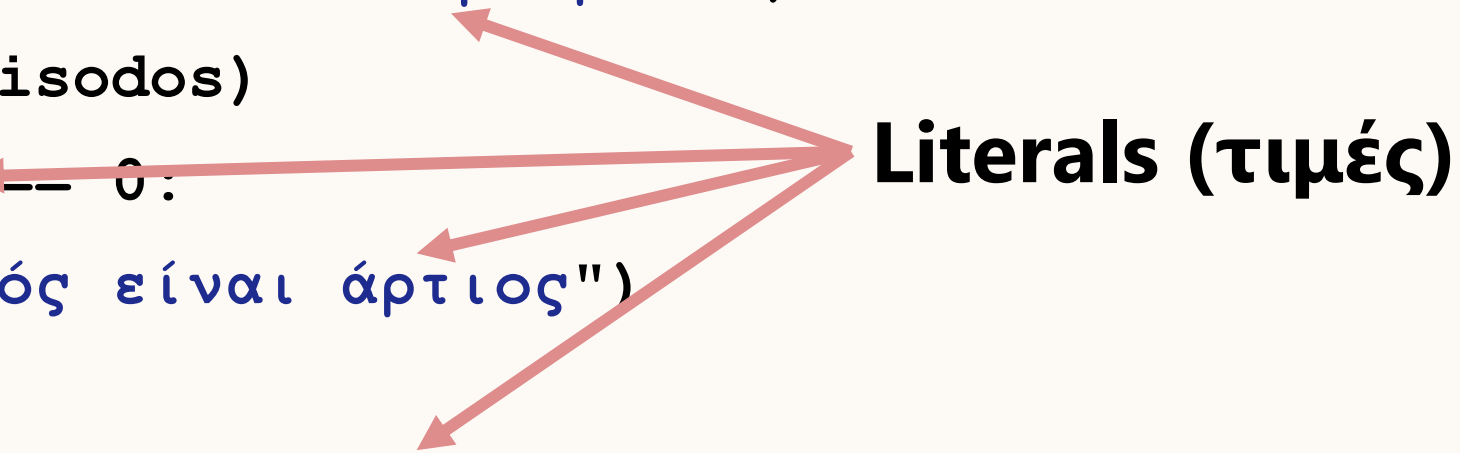
```
eisodos = input("Δώσε έναν αριθμό: ")
akeraios = int(eisodos)
if akeraios % 2 == 0:
  print("Ο αριθμός είναι άρτιος")
else:
  print("Ο αριθμός είναι περιττός")
```



Συναρτήσεις και Statements

Ένα πρόγραμμα σε Python

```
eisodos = input("Δώσε έναν αριθμό: ")
akeraios = int(eisodos)
if akeraios % 2 == 0:
  print("Ο αριθμός είναι άρτιος")
else
  print("Ο αριθμός είναι περιττός")
```



Λογικές πράξεις

x	not x
True	False
False	True

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

x	y	x or y
False	False	False
False	True	True
True	False	True
True	True	True

```
a = 4
print(not a==5)
```
```
True
```

```
a = 12
print(a<20 and a>12)
```
```
False
```

```
x = 5
y = 12
print(x>y or y<20)
```
```
True
```

Σειρά των τελεστών (order of operations)

```
a =  2 - 5 * 2  > 4
```

```
b =   2 + 5 * 3  > 10  and   4 + 6  == 11
```

Σειρά των τελεστών (order of operations)

```
a = (2 -(5 * 2))> 4
```

```
b = ((2 +(5 * 3))> 10) and ((4 + 6) == 11)
```

...γι΄ αυτό χρησιμοποιούμε παρενθέσεις!

Σειρά των τελεστών

1. Παρενθέσεις `()`
   - και συναρτήσεις
2. Πρόσημο `-, +`
3. Δύναμη `**`
4. Πολλαπλασιασμός, διαίρεση, mod, div `*, /, //, %`
5. Πρόσθεση, αφαίρεση `+, -` (διαφορετικές από το πρόσημο!)
6. Συγκρίσεις `==, !=, <=, >=, >, <`
7. `not`
8. `and`
9. `or`

Τελεστές με την ίδια προτεραιότητα αποτιμώνται **από τα αριστερά προς τα δεξιά**

Υπολογίζοντας εκφράσεις

```
10 + 6 / 2 * 3 - 4
```
= 15

```
2 ** 3 % 3 + 5 * 2
```
= 12

```
2 * (3 + 4) ** 2 - 10 / 2 + 7 % 3
```
= 94

```
(5 + 3) < 7 and abs(-10) == 10
```
= False

```
max(8, 6, 2) + 2 ** 3 > 15 and 7 % 3 == 1
```
= True

```
3 ** 3 > 10 or not 6 // 4 == 2 and abs(-5) == 0
```
= True

Πολλαπλή σύγκριση

Σε αντίθεση με παραδοσιακές γλώσσες, η Python επιτρέπει πολλαπλή σύγκριση:

- `a==b==c` ισοδυναμεί με `a==b and b==c`
- `a<b<c>d` ισοδυναμεί με `a<b and b<c and c>d`

Για τις ανάγκες του μαθήματος, μπορούμε να χρησιμοποιούμε όποια μορφή προτιμάμε

Δομές επιλογής

- Ένα πρόγραμμα που κάνει απλώς μια ακολουθία πράξεων δεν είναι πολύ χρήσιμο
- Η δομή επιλογής μας επιτρέπει να *διακλαδώνουμε* τον κώδικα με βάση κάποια συνθήκη

Δομή επιλογής: η εντολή `if`

- Η δομή επιλογής **`if`** λέει στον interpreter να εκτελέσει κάποιες από τις εντολές **μόνο αν** ισχύει κάποια συγκεκριμένη συνθήκη

Σύνταξη `if`

Προσοχή στην <u>εσοχή</u>! Παράδειγμα στην επόμενη διαφάνεια

```
if <συνθήκη>:
    <εντολές>
```

Άνω και κάτω τελεία στο τέλος της συνθήκης

- Η **<*συνθήκη*>** είναι μια <u>έκφραση</u> που όταν υπολογιστεί θα πρέπει να επιστρέφει μια τιμή **`bool`***
- Μπορεί να είναι μια σύγκριση, μια σύνθετη λογική πράξη, ή και μόνο μια μεταβλητή

*Αυτό δεν είναι ακριβώς αλήθεια. Θα επιστρέψουμε στο τέλος της διάλεξης.

Σύνταξη `if`: παράδειγμα

```
vathmos = int(input("Δώσε το βαθμό σου: "))
if vathmos>=5:
    print("Πέρασες με: ")
    print(vathmos)
print("Τέλος")
```

- Η εσοχή ορίζει το **μπλοκ εντολών** που υπάγεται στην **`if`**: οι δυο εντολές θα εκτελεστούν μόνο αν η συνθήκη είναι **`True`**
- Η τελευταία εντολή βρίσκεται εκτός του μπλοκ, οπότε δεν υπάγεται στην **`if`**. Θα εκτελείται σε κάθε ενδεχόμενο.

Σύνταξη `if-else`

```
if <συνθήκη>:
    <εντολές>
else:
    <άλλες εντολές>
```

- Οι *άλλες εντολές* εκτελούνται μόνο όταν η συνθήκη είναι **`False`**

Σύνταξη `if-else`: παράδειγμα

```
vathmos = int(input("Δώσε το βαθμό σου"))
if vathmos>=5:
  print("Πέρασες με:")
  print(vathmos)
else:
  print("Δεν πέρασες.")
  print("Ο βαθμός σου ήταν:", vathmos)
print("Τέλος")
```

- Το μπλοκ της **`else`** θα εκτελεστεί σε περίπτωση που η συνθήκη δεν ισχύει.

Παράδειγμα 1

Να γραφτεί κώδικας σε Python που ζητάει έναν αριθμό από το χρήστη. Αν ο αριθμός είναι θετικός, να εμφανίζει το μισό του. Αν όχι, να εμφανίζει το διπλάσιό του.

Παράδειγμα 1 - Λύση

Να γραφτεί κώδικας σε Python που ζητάει έναν αριθμό από το χρήστη. Αν ο αριθμός είναι θετικός, να εμφανίζει το μισό του. Αν όχι, να εμφανίζει το διπλάσιό του.

```
number = int(input("Δώσε έναν αριθμό: "))
if number>0:
  print(number/2)
else:
  print(number*2)
```

Μπλοκ εντολών 1/2

Στο προγραμματιστικό παράδειγμα που αποκαλούμε **Δομημένο Προγραμματισμό** έχουμε στη διάθεσή μας **δομές ελέγχου** (επιλογής και επανάληψης) που καθορίζουν αν και πόσες φορές θα εκτελεστούν κάποιες εντολές.

Οι εντολές αυτές συχνά είναι παραπάνω από μία, οπότε πρέπει να τις ομαδοποιούμε σε **μπλοκ** ώστε το πρόγραμμα να ξέρει ποιες είναι αυτές που πρέπει να εκτελεστούν υπό όρους.

Μπλοκ εντολών 2/2

Σε διαφορετικές γλώσσες τα μπλοκ οριοθετούνται με διαφορετικό τρόπο.

- Στη C, C++, C#, Java, τα μπλοκ οριοθετούνται με τα σύμβολα **{ }**
- Στην ψευδογλώσσα οριοθετούνται π.χ. με **`Αν…Τέλος_Αν`**
- Στην Python οριοθετούνται με **εσοχές**

Εσοχές

- Στις περισσότερες γλώσσες, οι εσοχές χρησιμεύουν για να διαβάζεται πιο εύκολα ο κώδικας
- Στην Python οι εσοχές είναι **απαραίτητο συστατικό της σύνταξης**

Η επίδραση των εσοχών

```
if a>1:
    if b<2:
        b+=3
    else:
        a+=2
```

```
if a>1:
    if b<2:
        b+=3
else:
    a+=2
```

Στην πρώτη περίπτωση ο κώδικας του **`else`** εκτελείται αν **η πρώτη** συνθήκη είναι ψευδής. Στη δεύτερη περίπτωση εκτελείται αν **η δεύτερη** συνθήκη είναι ψευδής.

Σύνταξη `if-elif-else`

```
if <συνθήκη>:
        <1ες εντολές>
elif <συνθήκη>:
        <2ες εντολές>
elif <συνθήκη>:
        <3ες κλπ εντολές>
…
else:
        <τελευταίες εντολές>
```

Η άνω και κάτω τελεία (:) ορίζει το τέλος της συνθήκης και την αρχή του αντίστοιχου μπλοκ εντολών

- Το **`elif`** είναι σύντμηση του **else if**
- Μόνο ένα μπλοκ εντολών εκτελείται σε κάθε εκτέλεση του προγράμματος

Παράδειγμα if-elif-else

```
a = int(input("Δώσε έναν αριθμό: "))
if a > 0:
    print("Θετικός.")
elif a < 0:
    print("Αρνητικός.")
else:
    print("Μηδέν.")
```

Παράδειγμα 2

Να γραφτεί κώδικας σε Python που θα δέχεται τα μήκη των τριών πλευρών ενός τριγώνου και θα ενημερώνει το χρήστη αν το τρίγωνο είναι:

1. Ισόπλευρο (όλες οι πλευρές ίσες)
2. Ισοσκελές (δύο πλευρές ίσες)
3. Σκαληνό (όλες οι πλευρές διαφορετικές)

Παράδειγμα 2 - Λύση

Να γραφτεί κώδικας σε Python που θα δέχεται τα μήκη των τριών πλευρών ενός τριγώνου και θα ενημερώνει το χρήστη αν το τρίγωνο είναι:

1. Ισόπλευρο (όλες οι πλευρές ίσες)
2. Ισοσκελές (δύο πλευρές ίσες)
3. Σκαληνό (όλες οι πλευρές διαφορετικές)

```
m1 = float (input("Μήκος πρώτης πλευράς: "))
m2 = float (input("Μήκος δεύτερης πλευράς: "))
m3 = float (input("Μήκος τρίτης πλευράς: "))
if m1==m2==m3:
  print("Το τρίγωνο είναι ισόπλευρο.")
elif m1==m2 or m2==m3 or m1==m3:
  print("Το τρίγωνο είναι ισοσκελές.")
else:
  print("Το τρίγωνο είναι σκαληνό.")
```

Εμφωλευμένες (φωλιασμένες) δομές

```
number = int(input("Δώσε έναν αριθμό: "))
if number > 10:
  if number > 100:
    print ("μεγαλύτερος του 100")
    print ("δηλαδή πολύ μεγάλος")
  else:
    print ("μεταξύ 10 και 100")
    print ("δηλαδή ενδιάμεσος")
else:
    print ("μικρότερος του 10")
```

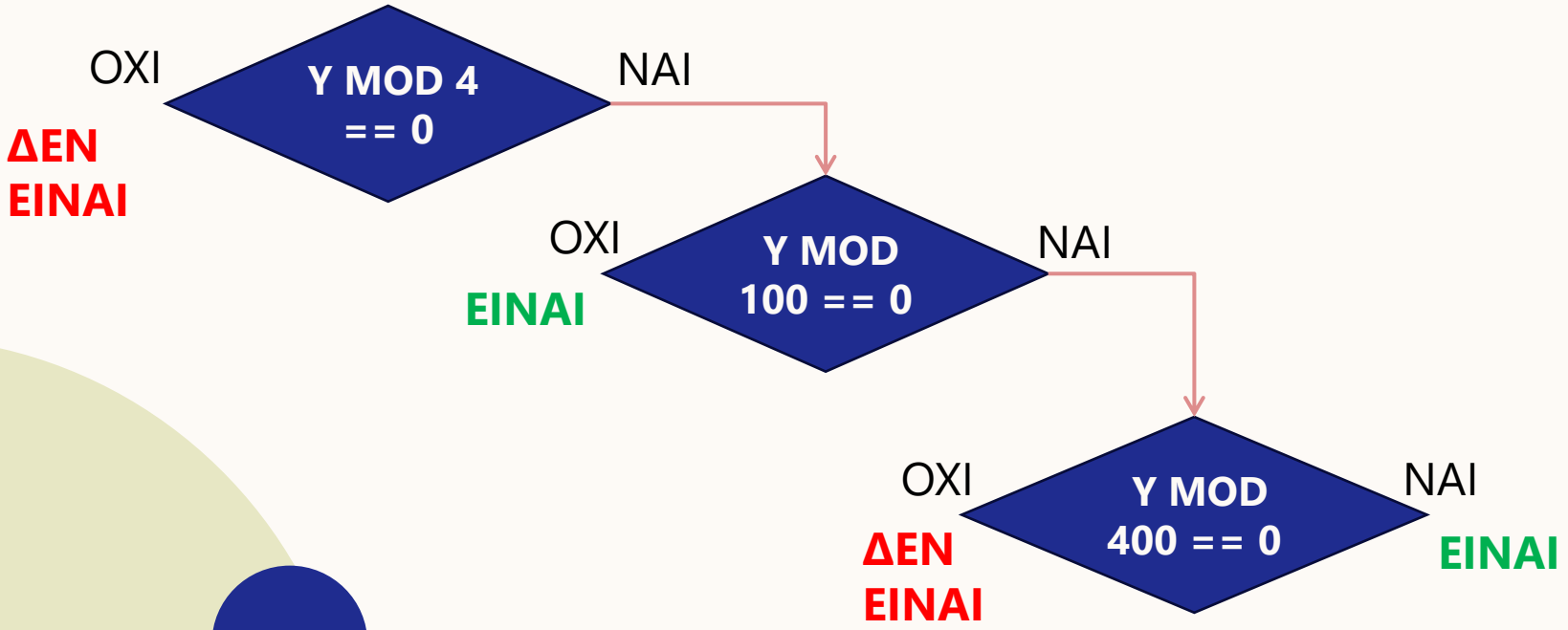
Αφού εντός του κάθε μπλοκ μπορούμε να βάλουμε **όσες** και **όποιες** εντολές θέλουμε, μπορούμε να βάλουμε και εντολές που δημιουργούν **δικά τους μπλοκ**

Παράδειγμα 3

Ένα έτος είναι δίσεκτο αν διαιρείται με το 4, εκτός αν διαιρείται με το 100 οπότε δεν είναι δίσεκτο, εκτός αν διαιρείται και με το 400, οπότε είναι δίσεκτο. Να γραφεί κώδικας σε Python που θα δέχεται ένα έτος από το χρήστη και θα ελέγχει αν είναι δίσεκτο ή όχι.

Παράδειγμα 3 – Οπτικοποίηση

Ένα έτος είναι δίσεκτο αν διαιρείται με το 4, εκτός αν διαιρείται με το 100 οπότε δεν είναι δίσεκτο, εκτός αν διαιρείται και με το 400, οπότε είναι δίσεκτο. Να γραφεί κώδικας σε Python που θα δέχεται ένα έτος από το χρήστη και θα ελέγχει αν είναι δίσεκτο ή όχι.



Παράδειγμα 3 - Λύση

Ένα έτος είναι δίσεκτο αν διαιρείται με το 4, εκτός αν διαιρείται με το 100 οπότε δεν είναι δίσεκτο, εκτός αν διαιρείται και με το 400, οπότε είναι δίσεκτο. Να γραφεί κώδικας σε Python που θα δέχεται ένα έτος από το χρήστη και θα ελέγχει αν είναι δίσεκτο ή όχι.

```
year = int(input("Δώσε το έτος: "))
if year % 4 != 0:
  print("Δεν είναι")
else:
  if year % 100 != 0:
    if year % 400 == 0:
      print("Είναι")
    else:
      print("Δεν είναι")
  else:
    print("Είναι")
```

Εναλλακτική δομή επιλογής: match-case

```
year=input("Τι έτος είσαι;")
match year:
  case "Πρώτο":
    print("Λάθος μάθημα;")
  case "Δεύτερο":
    print("Καλωσήρθες!")
  case y if y in ["Τρίτο", "Τέταρτο", "Πέμπτο",
                  "Έκτο", "Έβδομο"]:
    print("Καλή τύχη αυτή τη φορά!")
  case default:
    print("Άγνωστο έτος. Μήπως έκανες τυπογραφικό;")
```

Έλεγχος για παρουσία στοιχείου σε λίστα. Θα το δούμε στη Διάλεξη 4.

Παρότι έχει τις χρήσεις της, η δομή αυτή δε θα μας απασχολήσει άλλο σε αυτό το μάθημα.

Συνθήκες που δεν είναι `bool`

- Κατά κανόνα, χρησιμοποιούμε την εντολή **`if`** με μια έκφραση που επιστρέφει μια λογική τιμή
  - Αν η τιμή είναι **`True`**, ο κώδικας εκτελείται
- Μπορούμε όμως να το χρησιμοποιήσουμε και με εκφράσεις που επιστρέφουν άλλους τύπους δεδομένων
- Η **`if`** **δεν** εκτελεί το μπλοκ κώδικά της, αν η έκφρασή της επιστρέψει:
  - **`False`**
  - **`0`**
  - **`None`**
  - Άδεια συλλογή (π.χ. άδεια λίστα **`[]`**, συμβολοσειρά **`""`**, σύνολο **`{}`**, ή λεξικό **`{}`**)
- Σε οποιαδήποτε άλλη περίπτωση, το μπλοκ θα εκτελεστεί

False-like και True-like

False-like

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

True-like

- *Οτιδήποτε άλλο*

Τι θα εμφανίσει ο κώδικας;

```
a = 4
b = 5
if a-b:
  print("Ναι")
else:
  print("Όχι")
```

4 - 5 = -1
Είπαμε ότι μόνο το **0**, το **None** και η κενή συλλογή μετράνε για **False**
Άρα **True**, άρα τυπώνει **"Ναι"**

Τι θα εμφανίσει ο κώδικας;

```
a = 4
if not abs(a):
   print("Ναι")
else:
   print("Όχι")
```

abs(4) = 4
Αφού το **4** μετράει ως **True**,
το **not 4** θα μετράει για **False**
Άρα τυπώνει "Όχι".

Τι θα εμφανίσει ο κώδικας;

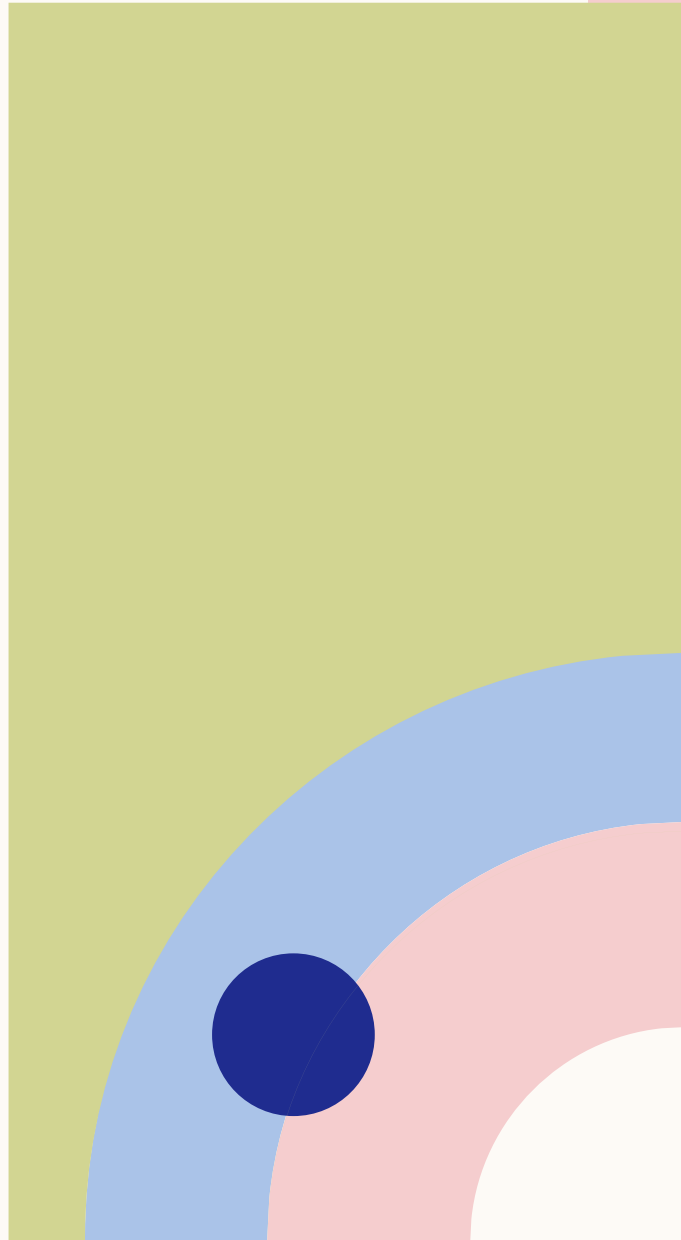
```
a = 4
if print(a):
   print("Ναι")
else:
   print("Όχι")
```

Όλες οι συναρτήσεις επιστρέφουν κάποια τιμή. Η **`print()`** επιστρέφει **`None`**, που μετράει για **`False`**
Άρα τυπώνει "Όχι".

Λογικές πράξεις με μη λογικούς τελεστές

Με τον ίδιο τρόπο που η **`if`** μπορεί να δεχτεί αντικείμενα που δεν είναι **`bool`**, έτσι και η **`and`** και η **`or`** μπορούν να λειτουργήσουν με τέτοια αντικείμενα

- **`A or B`**: Αν το **`A`** δεν είναι **`False`**, **`0`**, **`None`**, κενή συλλογή, τότε επιστρέφει το **`A`**. Αλλιώς επιστρέφει το **`B`**
- **`A and B`**: Αν το **`A`** είναι **`False`**, **`0`**, **`None`**, κενή συλλογή, επιστρέφει το **`A`**. Αλλιώς επιστρέφει το **`B`**
- **`not A:`** Αν το **`A`** είναι **`False`**, **`0`**, **`None`**, κενή συλλογή, επιστρέφει **`True`**. Αλλιώς, **`False`**.



Τι θα επιστρέψουν;

- `0 or 1` →`1`
- `[] and True` →`[]`
- `None or "Hello"` →`"Hello"`
- `100 or 5` →`100`
- `True and ""` →`""`
- `not 5` →`False`
- `False or True` →`True`
- `False and True` →`False`

Οπότε οι κλασικές λογικές πράξεις είναι υποκατηγορίες του γενικευμένου κανόνα!

Γιατί όμως;

Γιατί με το μηχανισμό αυτό, μπορούμε να γράφουμε πολύ αποτελεσματικό κώδικα

- Π.χ.: αν η θύρα της σύνδεσης είναι η 443, το πρωτόκολλο να είναι “https://”, αλλιώς να είναι “http://”

Λύση 1:

```
if port==443:
  protocol = "https://"
else:
  protocol = "http://"
```

Λύση 2:

```
protocol = port==443 and "https://" or "http://"
```

Εναλλακτική σύνταξη: `if` σε μία γραμμή

Αν η θύρα της σύνδεσης είναι η 443, το πρωτόκολλο να είναι “https://”, αλλιώς να είναι “http://”

Λύση 1:

```
if port==443:
  protocol = "https://"
else:
  protocol = "http://"
```

Λύση 3:

```
protocol = "https://" if port==443 else "http://"
```

Αντί να ελέγχει ένα μπλοκ εντολών, η **`if`** με αυτή τη σύνταξη επιστρέφει μια τιμή.
Πιο κοντά στην ανθρώπινη γλώσσα!

Too many options! 😵‍💫

- Οι επιλογές που προσφέρει η Python, όντας μια σύγχρονη γλώσσα προγραμματισμού, ενδέχεται να σας μοιάζουν χαοτικά πολλές
- Δε χρειάζεται να μπορούμε να προγραμματίσουμε με όλους αυτούς τους τρόπους ταυτόχρονα (αυτό το εξάμηνο...)
  - Ούτε θα σας ζητηθεί στις εξετάσεις να λύσετε ένα πρόβλημα χρησιμοποιώντας μια *συγκεκριμένη* δομή επιλογής
- Όμως θα πρέπει:
  1. Να γνωρίζουμε ότι αυτές οι δομές υπάρχουν, και να μπορούμε να κατανοήσουμε πώς θα συμπεριφερθεί ο κώδικας που τους περιέχει
  2. Να συνηθίσουμε την πιο βασική αρχή του προγραμματισμού: **κάθε εντολή συμπεριφέρεται με πολύ συγκεκριμένο τρόπο, που αν τον ακολουθήσουμε κατά γράμμα, μπορούμε να προβλέψουμε το αποτέλεσμά της**

Παράδειγμα 4 (για να επιστρέψουμε στα βασικά...)

Ένα εισιτήριο σινεμά κοστολογείται ως εξής:

- Ενήλικες κάτω των 65: **12**
- Ενήλικες άνω των 65: **10**
- Παιδιά άνω των 3: **8**
- Παιδιά κάτω των 3: **δωρεάν**

Εκπτώσεις:

- Φοιτητές/μαθητές: **-2**
- Κάρτα μέλους: **-2**
- Και τα δύο μαζί: **-3**

Να γραφεί κώδικας που να ρωτάει τα στοιχεία ενός ατόμου και να επιστρέφει το τελικό κόστος του εισιτηρίου του.

Παράδειγμα 4

Ένα εισιτήριο σινεμά κοστολογείται ως εξής:

- Ενήλικες κάτω των 65: **12**
- Ενήλικες άνω των 65: **10**
- Παιδιά άνω των 3: **8**
- Παιδιά κάτω των 3: **δωρεάν**

Εκπτώσεις:

- Φοιτητές/μαθητές: **-2**
- Κάρτα μέλους: **-2**
- Και τα δύο μαζί: **-3**

Να γραφεί κώδικας που να ρωτάει τα στοιχεία ενός ατόμου και να επιστρέφει το τελικό κόστος του εισιτηρίου του.

```
age = int(input("δώσε
    ηλικία"))
paso = input("Έχεις
    πάσο; (ναι/όχι)")
melos = input("Είσαι
    μέλος; (ναι/όχι)")


if age > 65:
  base_price = 10
elif age >=18:
  base_price = 12
elif age >=3:
  base_price = 8
else:
  base_price = 0
```

```
if paso == "ναι":
  if melos == "ναι":
    discount = 3
  else:
    discount = 2
else:
  if melos == "ναι":
    discount = 2
  else:
    discount = 0

print("Η τιμή σου είναι:",
    base_price - discount)
```

Σφάλματα

Τύποι και διαχείριση

Τρεις τύποι σφαλμάτων

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

Syntax errors

Παραβιάσεις των συντακτικών κανόνων της γλώσσας

- Π.χ.:

```
a = int(input("Δώσε το βαθμό σου: "))
if a>=5
  print("Πέρασες")
```

Λείπει το :

```
a = int(input("Δώσε το βαθμό σου: "))
if a=10:
  print("Άριστα!")
```

= αντί για ==

Διαχείριση συντακτικών σφαλμάτων

- Η Python αρνείται να εκτελέσει κώδικα με συντακτικά σφάλματα
  - Βγάζει Error που επισημαίνει τη γραμμή και τον τύπο του σφάλματος (και πιθανές διορθώσεις)

```
if a>=5
       ^
SyntaxError: expected ':'
```

```
if a=10:
    ^
SyntaxError: invalid syntax. Maybe you
meant '==' or ':=' instead of '='?
```

Logical errors

Λάθη στο σχεδιασμό του αλγορίθμου, τα οποία δεν προκαλούν διακοπή του προγράμματος αλλά δίνουν ανεπιθύμητα αποτελέσματα

- Π.χ.:

```
a = int(input("Δώσε το βαθμό σου: "))
if a>=5:
  print("Κόπηκες")
else:
  print("Πέρασες")
```

Διαχείριση λογικών σφαλμάτων

- Τα Logical errors είναι πολύ πιο επικίνδυνα από τα Syntax errors, γιατί ο κώδικας συχνά *μοιάζει* να λειτουργεί κανονικά
- Ενδέχεται ακόμα και το πρόγραμμα να βγει στην αγορά χωρίς να τα έχουμε εντοπίσει
- Μόνη λύση: **Testing**

Testing

- Πριν παραδώσουμε τον κώδικά μας, **τον δοκιμάζουμε με διαφορετικά σενάρια**
  - Δίνουμε διαφορετικά *inputs*, από τα οποία περιμένουμε διαφορετικά *outputs*, και επιβεβαιώνουμε τα αποτελέσματα
- Σε επαγγελματικά περιβάλλοντα, εφαρμόζουμε **Unit Testing**
  - Για κάθε τμήμα κώδικα, γράφουμε επιπλέον κώδικα που
    1. Τον τροφοδοτεί με διάφορες εισόδους
    2. Ελέγχει τις εξόδους και τις συγκρίνει με τις αναμενόμενες
    3. Μας ενημερώνει αν υπάρξει απόκλιση
  - Στη συνέχεια, αν τροποποιήσουμε ένα κομμάτι κώδικα (π.χ. για να το κάνουμε πιο γρήγορο, ή πιο ευανάγνωστο), ξανα-τρέχουμε τα Unit Tests του
    - Δηλαδή αυτοματοποιούμε τη διαδικασία του ελέγχου

Runtime Errors / Exceptions

- Τα σφάλματα χρόνου εκτέλεσης προκύπτουν κατά την εκτέλεση του προγράμματος
  - Σε αντίθεση με τα λογικά λάθη, που μπορεί να περάσουν απαρατήρητα, τα Exceptions προκαλούν διακοπή του προγράμματος και μήνυμα λάθους
- Πιθανά σενάρια:
  - Εντολές που είναι συντακτικά σωστές αλλά οδηγούν σε σφάλμα
  - Ενέργειες του χρήστη που δεν έχουμε προβλέψει
  - Άλλα απρόβλεπτα εξωτερικά συμβάντα (σφάλμα δίσκου, σύνδεσης κλπ)

Παραδείγματα Exceptions

```
a = 5
b = "hello"
c = a + b
```

```
----> 3 c = a + b

TypeError: unsupported
operand type(s) for +:
'int' and 'str'
```

```
d = int(input("Δώσε
        παρονομαστή: "))
print(1/d)
```

```
Δώσε παρονομαστή: 0
--------------------
ZeroDivisionError
----> 2 print(1/d)

ZeroDivisionError:
division by zero
```

```
d = int(input("Δώσε
        παρονομαστή: "))
print(1/d)
```

```
Δώσε παρονομαστή: Hello
-----------------------
ValueError
----> 1 d = int(input("Δώσε
                παρονομαστή: "))
      2 print(1/d)

ValueError: invalid literal
for int() with base 10:
'Hello'
```

Διαχείριση Exceptions

- Η πρώτη γραμμή διαχείρισης είναι προφανής: **Testing**
- Όμως, δε μπορούμε να δοκιμάσουμε κάθε ενδεχόμενο
  - Απρόβλεπτη συμπεριφορά χρήστη
  - Αβεβαιότητα στις online εφαρμογές (αργή σύνδεση, πεσμένος server…)
  - Άλλα απρόβλεπτα συμβάντα
- Λύση: **Exception handling**

Δομή `try...except`

- *Δοκιμάζει* να τρέξει ένα τμήμα κώδικα
  - Αν όλα πάνε καλά, συνεχίζει
  - Αν προκληθεί Exception, ενεργοποιεί ένα άλλο τμήμα κώδικα

Σύνταξη:

```
try:
  <Κώδικας που μπορεί να προκαλέσει σφάλμα>
except:
  <Διαχείριση σφάλματος>
```

Παράδειγμα `try...except`

```
a = input("Δώσε παρονομαστή: ")
try:
  denom = float(a)
  fraction = 1/denom
except:
  fraction = None
  print("Κάτι πήγε λάθος")
print("Το αποτέλεσμα είναι",fraction)
```

```
Δώσε παρονομαστή: 0
Κάτι πήγε λάθος
Το αποτέλεσμα είναι None
```

- Πλεονέκτημα: η εκτέλεση δε διακόπηκε
  - Διαχειριστήκαμε το σφάλμα «με χάρη» (gracefully)
- Μειονέκτημα: δε ξέρουμε τι πήγε λάθος
  - Τα μηνύματα σφάλματος είναι χρήσιμα...

Βελτιωμένο try…except

```
a = input("Δώσε παρονομαστή: ")
try:
  denom = float(a)
  fraction = 1/denom
except Exception as e:
  fraction = None
  print("Σφάλμα. Τύπος σφάλματος:",
  type(e), ". Μήνυμα: ", e)
print("Το αποτέλεσμα είναι",fraction)
```

```
Δώσε παρονομαστή: 0
Σφάλμα. Τύπος σφάλματος:
<class 'ZeroDivisionError'>.
Μήνυμα:  float division by zero
Το αποτέλεσμα είναι None
```

- Το e είναι ένα αντικείμενο/μεταβλητή
- Τι τύπου;
- Μέσα του περιέχει πληροφορίες για το σφάλμα, για να μας βοηθήσει στην αποσφαλμάτωση
  - Χωρίς να διακόψουμε την εκτέλεση του προγράμματος

try…except για συγκεκριμένους τύπους σφαλμάτων

```
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
print("Το αποτέλεσμα είναι",fraction)
```

```
Δώσε παρονομαστή: c
Παρακαλώ δώστε αριθμό, όχι γράμματα.
Το αποτέλεσμα είναι None
```

- Το e είναι ένα αντικείμενο/μεταβλητή
- Τι τύπου;
- Μέσα του περιέχει πληροφορίες για το σφάλμα, για να μας βοηθήσει στην αποσφαλμάτωση
  - Χωρίς να διακόψουμε την εκτέλεση του προγράμματος

Δομή try…except…finally

```
a = input("Δώσε παρονομαστή: ")
try:
  denom = float(a)
  fraction = 1/denom
except Exception as e:
  fraction = None
  print("Κάτι πήγε λάθος.
                Το μήνυμα ήταν:", e)
finally:
  print("Το αποτέλεσμα είναι",fraction)

Δώσε παρονομαστή: 0
Κάτι πήγε λάθος. Το μήνυμα
     ήταν: float division by zero
Το αποτέλεσμα είναι None
```

- Το μπλοκ της **finally** εκτελείται **παντα**
- Ποια η διαφορά με το να έχουμε τον κώδικα εκτός των μπλοκ όπως πριν;
  - Το μπλοκ **finally** εκτελείται και όταν
    - Προκύψει exception που δεν το συλλάβαμε (και άρα η εκτέλεση θα τερματίσει)
    - Είπαμε εμείς στο πρόγραμμα να σταματήσει (με εντολές **break**, **return**, **continue** που θα δούμε σε άλλη διάλεξη)

Εντολή raise

```
number = float(input("Δώσε
      έναν θετικό αριθμό: "))
if number <= 0:
   raise ValueError("Είπαμε
      θετικό!")
print(pow(number,0.5))
```

- "Προκαλούμε" ένα exception
- Μπορούμε να το συλλάβουμε με ένα **try**...**except** και να το διαχειριστούμε

Παράδειγμα 5

Να γραφεί κώδικας που θα δέχεται δυο αριθμούς **x** και **y**, και θα εμφανίζει την **y**-οστή ρίζα του **x**.

Αν η ρίζα δε μπορεί να υπολογιστεί (*ως πραγματικός αριθμός*) για οποιοδήποτε λόγο, να εμφανίζεται σχετικό μήνυμα προς το χρήστη.

Παράδειγμα 5

Να γραφεί κώδικας που θα δέχεται δυο αριθμούς **x** και **y**, και θα εμφανίζει την **y**-οστή ρίζα του **x**.

Αν η ρίζα δε μπορεί να υπολογιστεί (*ως πραγματικός αριθμός*) για οποιοδήποτε λόγο, να εμφανίζεται σχετικό μήνυμα προς το χρήστη.

```
try:
  x = float(input("Δώσε βάση: "))
  y = float(input("Δώσε εκθέτη: "))
  if (y<0):
    print("Όχι αρνητική ρίζα")
  else:
    print(pow(x,1/y))
except ValueError as e:
  print("Αριθμό, όχι κείμενο!")
except ZeroDivisionError as e:
  print("Όχι μηδενική ρίζα")
except Exception as e:
  print("Άγνωστο σφάλμα:",type(e), e)
```