Μεταβλητές

Όταν γράφουμε a=3 τότε αποθηκεύουμε στη μνήμη του υπολογιστή την τιμή 3. Μπορούμε όποτε θέλουμε μετά να αναφερθούμε σε αυτή την τιμή χρησμοποιώντας το όνομα της μεταβλητής

In [1]:
```
a=3
```

In [2]:
```
a
```

Out[2]:
```
3
```

In [3]:
```
a+a
```

Out[3]:
```
6
```

In [4]:
```
a = 'mitsos'
```

In [5]:
```
a
```

Out[5]:
```
'mitsos'
```

Μπορούμε να κάνουμε διάφορες πράξεις μεταξύ μεταβλητών:

In [6]:
```
a=3
b=4
print (a+b)
```

```
7
```

Τα κενα δεν έχουν σημασία (αρκεί όλες οι γραμμές να ξεκινάνε από το ίδιο κενό).

Όλα τα παρακάτω είναι ισοδύναμα:

In [8]:
```
a=3
a = 3
a                                =
```

Στα παρακάτω όμως υπάρχει λάθος!

In [9]:
```
a=3
 a=3 # Αυτό ξεκινάει με ένα κενό πιο μέσα!
```

```
  File "<ipython-input-9-fc87d32c6889>", line 2
    a=3 # Αυτό ξεκινάει με ένα κενό πιο μέσα!
    ^
IndentationError: unexpected indent
```

Με την print μπορούμε να τυπώνουμε τις τιμές πολλών μεταβλητών:

In [10]:
```
a="the answer is"
b=7
```

In [11]:
```
print (a,b)
```

```
the answer is 7
```

In [12]:
```
print ("this answer is", b)
```

```
this answer is 7
```

Αν μέσα σε ένα string βάλουμε το `{}` τότε μπορούμε να βάλουμε μία μεταβλητή σε αυτή τη θέση του string. Για να το κάνουμε αυτό χρησιμοποιούμε τη μέθοδο `format` .

In [13]:
```
c = "answer is {}".format(b)
print (c)
```

```
answer is 7
```

Μπορούμε να βάλουμε πάνω από ένα `{}` σε ένα string:

In [14]:
```
a = 'James'
b = 'Bond'
print ('My name is {}. {} {}.'.format(b, a, b))
```

```
My name is Bond. James Bond.
```

Προσοχή στη διαφορά μεταξύ `=` και `==` :

In [29]:
```
a = 3 # Η μεταβλητή a παίρνει την τιμή 3
a == 3 # Ελέγχουμε αν η τιμή της μεταβλητής a είναι 3
```

Out[29]:
```
True
```

Η πράξη: +=

Έστω ότι έχουμε μία μεταβλητή που έχει την τιμή 3:

In [15]:
```
a = 3
print (a)
```

```
3
```

Πως μπορούμε να της αυξήσουμε τη τιμή κατά 1;

In [16]:
```
a = a + 1
print (a)
```

```
4
```

Το `a=a+1` χρησημοποιείται πολύ συχνά (στην ουσία κάθε φορά που "μετράμε" κάτι). Οπότε μπορούμε να το γράψουμε και σαν: `a += 1` . Παρομοίως μπορούμε να γράψουμε και `a += 4`

In [17]:
```
a += 4 # a = a + 4
print (a)
```

```
8
```

Το ίδιο μπορεί να γίνει με όλες τις άλλες πράξεις. Π.χ. το `a -= 1` είναι ισοδύναμε με `a = a - 1` .

In [18]:
```
a -= 1
print (a)
```

```
7
```

Functions (συναρτήσεις)

Οι συναρτήσεις είναι ένα τεράστιο κομμάτι της θεωρίας υπολογιστών. Μέσω των συναρτήσεων μπορούμε να "σπάσουμε" τον κώδικά μας σε μικρα λειτουργικά κομμάτια και να τον κάνουμε πιο οργανωμένο και πιο επαναχρησιμοποιήσουμε. Π.χ. μία συνάρτηση που υπολογίζει τον μέσο όρο μπορούμε να τη χρησιμοποιήσουμε σε πολλά σημεία του κώδικά μας. Στην python ορίζουμε μία συνάρτηση μέσω της `def` :

In [42]:
```
def f():
    print ('hello')

f()
```

```
hello
```

Μία συνάρτηση μπορεί να "επιστρέφει" μία τιμή σε αυτόν που την κάλεσε:

In [43]:
```
def f():
    print ('hello')
    return 4
a=f()
print (a)
```

```
hello
4
```

Μία συνάρτηση μπορεί να παίρνει καμία, μία ή παραπάνω παραμέτρους:

In [44]:
```
def f(t):
    return t+1
a=f(3)
print (a)
```

```
4
```

In [45]:
```
def f(q,w,e,r,t):
    return q+w-e/r*t
a=f(2,3,4,5,6)
print (a)
```

```
0.1999999999999993
```

Επίσης μία συνάρτηση μπορεί να έχει παραμέτρους με προκαθορισμένες τιμές. Αν η συνάρτηση κληθεί χωρίς να δοθεί μία τιμή σε αυτές τις παραμέτρους τότε χρησιμοποιείται η προκαθορισμένη τιμή:

In [46]:
```
def f(a,b=4):
    return a+b
```

In [47]:
```
f(2,3)
```

Out[47]:
```
5
```

In [48]:
```
f(2)
```

Out[48]:
```
6
```

Μία συνάρτηση μπορεί να έχει πολλές παραμέτρους με προκαθοριμένες τιμές:

In [49]:
```
def f(a,b=2,c=4):
    return a+b+C
```

Πρέπει όμως όλες αυτές οι παράμετροι να είναι δηλωμένες μετά από τις παραμέτρους χωρίς προκαθορισμένες τιμές:

In [50]:
```
def f(a,b=2,c):
    return 42
```

```
  File "<ipython-input-50-7943a91cece0>", line 1
    def f(a,b=2,c):
            ^
SyntaxError: non-default argument follows default argument
```

Προσοχή! όταν μεταβάλουμε ένα όρισμα της συνάρτησης, τότε αν αυτό το όρισμα είναι string, int, float ή bool (αυτοί οι τύποι λέγονται primitive) τότε αυτή η μεταβολή δεν φαίνεται από εκεί που καλέσαμε τη συνάρτηση:

In [53]:
```
def f(a):
    a = a + 1 # Αλλάζουμε την a

a=4
f(a)
print (a) # Η a δεν άλλαξε!
```

```
4
```

Οι συναρτήσεις δεν μπορούν να "δουν" τα primitive data types (int, string, bool) που έχουν οριστεί έξω από αυτές:

In [54]:
```
a=4
def f():
    a=5
f()
print (a)
```

```
4
```

Για να μπορέσει να "δει" μία συνάρτηση ένα primitive data type που έχει οριστεί έξω από αυτή, μπορούμε να χρησιμοποιήσουμε τη `global` :

Προσοχή Υπάρχει μια εξαιρετικά ενοχλητική ομάδα προγραμματιστών που θεωρεί ότι ποτέ δεν πρέπει να χρησιμοποιούμε τη global. Αγνοήστε τους.

In [55]:
```
a=4
def f():
    global a
    a=5
f()
print (a)
```

```
5
```

Προσοχή! οτιδήποτε υπάρχει "κάτω" από τη `return` αγνοείται:

In [56]:
```
def f():
    print ("hello")
    return 5
    print ("dsfgsdfg")

f()
```

```
hello
```

Out[56]: 5

Μία συνάρτηση που δεν επιστρέφει τίποτα επιστρέφει μία τιμή που είναι `None` .

In [57]:
```
def f():
    print ("hello")

print (f())
```

```
hello
None
```

Η `None` είναι ένας νέος τύπος δεδομένου:

In [58]:
```
type(None)
```

Out[58]: NoneType

Μία συνάρτηση μπορεί να περιέχει μία άλλη συνάρτηση:

In [59]:
```
def f(r):
    def g():
        return r + 5
    return g() + 3

f(1)
```

Out[59]: 9

Οι συναρτήσεις είναι και αυτές μεταβλητές τύπου `function` :

In [60]:
```
type(f)
```

Out[60]: function

Μπορούμε να ελέγξουμε αν μία μεταβλητή είναι συνάρτηση με τη `callable` :

In [61]:
```
callable(f)
```

Out[61]: True

Η εντολή `if`

Η εντολή `if` (εάν) εκτελεί άλλες εντολές ανάλογα με το αν μια έκφραση είναι `True` ή `False` .

In [20]:
```
if True:
    print ("Hello")
```

```
Hello
```

In [21]:
```
if False:
    print ("Hello") # Δεν εκτελείται
```

In [22]:
```
if 1<3:
    print ("Hello")
```

```
Hello
```

In [23]:
```
if 3<1 in [1,2]:
    print ("Hello") # Δεν εκτελείται
```

Προσοχή! όλα τα strings εκτός από το άδειο είναι `True` :

In [24]:
```
if 'mitsos':
    print ("hello")
```

```
hello
```

In [25]:
```
if '':
    print ("hello") # Δεν εκτελείται
```

Όλοι αριθμοί εκτός από το 0 είναι `True`

In [26]:
```
if 3453:
    print ("Hello")
```

```
Hello
```

In [27]:
```
if 0: # Αυτό είναι False
    print ("Hello")
```

In [28]:
```
if 0.000000000001: # Αυτό είναι True!!
    print ("Hello")
```

```
Hello
```

Αυτό δεν επιτρέπεται:

In [30]:
```
if a = 3:
    print ("Hello")
```

```
  File "<ipython-input-30-9d66b7bd4d9a>", line 1
    if a = 3:
         ^
SyntaxError: invalid syntax
```

Αυτό επιτρέπεται:

In [31]:
```
if a == 3:
    print ("Hello")
```

```
Hello
```

Μία if μπορεί να έχει "μέσα της" και άλλες if..

In [32]:
```
print ('1')
if True:
    print ('hello')
    if True:
        print ('Kostas')

print ('2')
```

```
1
hello
Kostas
2
```

Αν για κάποιο λόγο δεν θέλουμε να κάνει τίποτα η if, τότε πρέπει να χρησιμοποιήσουμε τη `pass`

In [33]:
```
print ('1')
if True:
    pass
print ('2')
```

```
1
2
```

Μπορούμε να δηλώσουμε τι θέλουμε να γίνεται όταν η συνθήκη της if ΔΕΝ είναι αληθής με την else:

In [34]:
```
print ('1')
if True:
    print ("hello") # <-- Μπαίνει εδώ
else:
    print ('world') # <-- Δεν μπαίνει εδώ
print ('2')
```

```
1
hello
2
```

In [35]:
```
print ('1')
if False:
    print ("hello") # <-- Δεν μπαίνει εδώ
else:
    print ('world') # <-- Μπαίνει εδώ
print ('2')
```

```
1
world
2
```

Επίσης μπορούμε να δηλώσουμε πολλές συνθήκες με την elif. Η python τις ελέγχει μία-μία και μόλις (και εάν) βρει τη πρώτη αληθή, τότε μπαίνει στο indentation.

In [36]:
```
print ('hello')
a=2
if a==1:
    print ('1')
elif a==2:
    print ('2')
else:
    print ('3')
print ('world')
```

```
hello
2
world
```

Δεν είναι απαραίτητο να υπάρχει η else

In [37]:
```
print ('hello')
a=3
if a==1:
    print ('1')
elif a==2:
    print ('2')
print ('world')
```

```
hello
world
```

Στην `if` μπορούμε να δηλώσουμε παραπάνω από μία συνθήκες ή να να χρησιμοποιήσουμε την `else` για να δηλώσουμε τι να γίνει αν όλες οι συνθήκες στις `if` και `elif` είναι `False`

In [38]:
```
age = 23
if age<18:
    status = 'ανήλικος'
else:
    status = 'ενήλικος'

print (status)
```

```
ενήλικος
```

Προσέχτε ότι το παραπάνω δεν ελέγχει τη περίπτωση λάθους:

In [39]:
```
age = -4

if age<18:
    status = 'ανήλικος'
else:
    status = 'ενήλικος'

print (status)
```

```
ανήλικος
```

Μπορούμε να βάλουμε πολλές `elif` ώστε να ελέγχουμε για πολλά ενδεχόμενα:

In [40]:
```
age = 50
if age <0:
    status = "λάθος. Αρνητική τιμή"
elif age < 18:
    status = "ανήλικος"
elif age < 120:
    status = "ενήλικος"
else:
    status = "λάθος. Υπερβολικά μεγάλη τιμή"
print (status)
```

```
ενήλικος
```

Δοκιμάστε το παραπάνω για διάφορες τιμές του `age` .

Επίσης, δεν είναι απαραίτητο να χρησιμοποιήσουμε την `else`:

In [41]:
```
age = 150
if age <0:
    status = "λάθος. Αρνητική τιμή"
elif age < 18:
    status = "ανήλικος"
elif age < 120:
    status = "ενήλικος"
elif age >= 120:
    status = "λάθος. Υπερβολικά μεγάλη τιμή"
print (status)
```

```
λάθος. Υπερβολικά μεγάλη τιμή
```

In [ ]:
```

```