

Περισσότερα σχετικά με συναρτήσεις

Συναρτήσεις στην Python – dive in

Έχουμε ορίσει και χρησιμοποιήσιμες δικές μας συναρτήσεις στην Python

Η γενική σύνταξη όπως έχουμε δει είναι:

```
def FunctionName( parameters ) :  
    block entolwn  
    return
```

Η ερώτηση είναι «τι ακριβώς περνούμε στον κώδικα της συνάρτησης μέσω των παραμέτρων?»

Όλες οι παράμετροι (ορίσματα) που χρησιμοποιούνται στην Python περνούν με αναφορά (**by reference**) στη διεύθυνση της μνήμης του υπολογιστή.

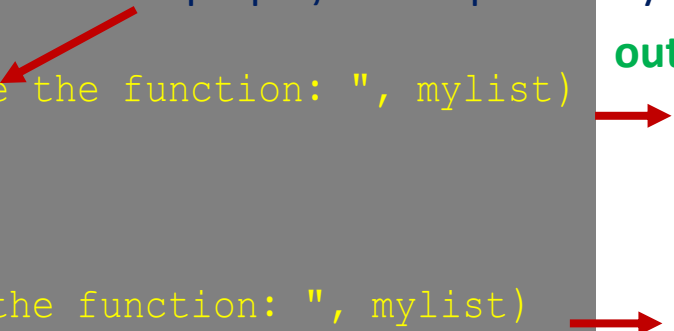
Αυτό σημαίνει ότι όταν αλλάξουμε την τιμή που είναι αποθηκευμένη στην διεύθυνση της μνήμης που αναφέρεται η παράμετρος τότε η αλλαγή αυτή θα εμφανιστεί και στο τμήμα του κώδικα που καλεί την συνάρτηση.

```
#!/usr/bin/python  
def changeme( mylist ):  
    mylist.append([1,2,3,4])  
    print("Values inside the function: ", mylist) → [10,20,30,1,2,3,4]  
    return  
mylist = [10,20,30]  
changeme( mylist )  
print("Values outside the function: ", mylist) → [10,20,30,1,2,3,4]
```

output

Συναρτήσεις στην Python – dive in

Η αναφορά στη διεύθυνση της μνήμης μπορεί να αλλάξει μέσα στο κύριο σώμα της συνάρτησης και να απενεξαρτοποιηθεί από τη διεύθυνση που καθόρισε το τμήμα του κώδικα που κάλεσε τη συνάρτηση:



```
def changeme( myList ):
    myList = [1,2,3,4]
    print("Values inside the function: ", myList)
    return
mylist = [10,20,30]
changeme( myList )
print("Values outside the function: ", myList)
```

Ορισμός αντικειμένου myList δημιουργεί νέα διεύθυνση

output

[1,2,3,4]

[10,20,30]

Στο παραπάνω παράδειγμα, η myList είναι **τοπική παράμετρος** (local variable list) για τη συνάρτηση και οι τιμές που εισαγάγουμε αλλάζουν την τοπική list αλλά δεν αλλάζουν τη λίστα που εισάγεται από το τμήμα που καλεί τη συνάρτηση.

```
def swap(a,b):
    temp = b
    b = a
    a = temp
    return
x = 2
y = 3
swap(x,y)
print("x, y", x,y)
```

output

2, 3

Ορίσματα συνάρτησης: Όρισμα με default τιμή

Μπορούμε να ορίσουμε μια συνάρτηση που κάποιο ή όλα τα ορίσματά της να θεωρούν μια συγκεκριμένη τιμή όταν η κλίση της συνάρτησης δεν παρέχει κάποια τιμή για το συγκεκριμένο όρισμα.

Παράδειγμα με default τιμή ορίσματος

```
def myfunc(x,y=50):  
    print("x=",x)  
    print("y=",y)  
  
myfunc(10) # klisi tis synartisis me ena orisma
```

- **Σημειωτέον:** Σε μια συνάρτηση μπορούμε να έχουμε όσο ορίσματα επιθυμούμε με ως ορίσματα με default τιμή. Ωστόσο από τη στιγμή που ένα όρισμα έχει τεθεί ως όρισμα με default τιμή, όλα τα ορίσματα που ακολουθούν στα δεξιά του θα πρέπει αναγκαστικά να είναι οπρ'

Ορίσματα συνάρτησης: Όρισμα με keywords

Στην περίπτωση αυτή ο χρήστης μπορεί να ορίσει το όνομα του ορίσματος με κάποιο Νούμερο. Αυτό δίνει την ευκολία στον χρήστη να μη χρησιμοποιήσει την ακριβή σειρά με την οποία χρησιμοποιούνται τα ορίσματα.

Παράδειγμα χρήσης ορισμάτων με keywords

```
def student(firstname, lastname):  
    print(firstname,lastname)
```

```
student(firstname='Fotis', lastname='Ptochos')  
student(lastname='Ptochos',firstname='Fotis')
```

output

[Fotis Ptochos]

[Fotis Ptochos]

Ορίσματα συνάρτησης: Μεταβλητός αριθμός ορισμάτων

Η συνάρτηση μπορεί να έχει μεταβλητό αριθμό ορισμάτων ανεξάρτητα αν τα ορίσματα είναι ορίσματα με default τιμή ή ορίσματα με keywords.

Ωστόσο χρησιμοποιούμε το σύμβολο ***args** για να περάσουμε ορίσματα που **δεν είναι** ορίσματα με keywords και το σύμβολο ****kwargs** για ορίσματα με keywords

- Η σύνταξη με ***args** σε συνάρτηση:
 - ✓ Χρησιμοποιεί το σύμβολο ***** για να δηλώσει ένα μεταβλητό αριθμό ορισμάτων. Συνήθως χρησιμοποιείται με το όνομα args
 - ✓ Επιτρέπει να ληφθούν περισσότερα ορίσματα από το πλήθος των ορισμάτων που ορίστηκαν επίσημα προηγουμένως. Με το ***** μπορούν να προστεθούν όσο ορίσματα χρειάζονται στα αρχικά επίσημα ορισμένα ορίσματα
 - ✓ Η μεταβλητή που συνδέουμε με το σύμβολο ***** παίρνει τη μορφή ακολουθίας και μπορούμε να την χρησιμοποιήσουμε όπως όλες αυτές τις μεταβλητές (όπως string, list, tuple) σε loops ή να τη χρησιμοποιήσουμε σε συναρτήσεις όπως η filter ή η loop.
- Η σύνταξη με ****kwargs** :
 - ✓ Χρησιμοποιείται για να περάσει μια λίστα μεταβλητού αριθμού ορισμάτων με keywords
 - ✓ Κάποιος θα μπορούσε να θεωρήσει τα kwargs σαν ένα dictionary το οποίο αντιστοιχεί κάθε keyword στο όρισμα που προσπαθούμε να περάσουμε. Για το λόγο αυτό όταν εξετάζουμε την ακολουθία δεν παίρνουμε τα Keywords με την σωστή σειρά

Παραδείγματα χρήσης συνάρτησης με *args

Παράδειγμα χρήσης ορισμάτων με *args

```
def myfunc(*argv):  
    for arg in argv:  
        print(arg)  
Myfunc('Hello', 'Welcome', 'to', 'Cyprus')
```

Output

Hello
Welcome
to
Cyprus

Παράδειγμα χρήσης ορισμάτων με *args

και 1 ειπλέον όρισμα

```
def myFun(arg1, *argv):  
    print ("First argument :", arg1)  
    for arg in argv:  
        print("Next argument through *argv :", arg)  
  
myFun('Hello', 'Welcome', 'to', 'GreeksforGreeks')
```

Output

First argument : Hello
Next argument through *argv : Welcome
Next argument through *argv : to
Next argument through *argv : GreeksforGreeks

Παραδείγματα χρήσης συνάρτησης με ****kwargs**

```
# Παράδειγμα **kwargs για μεταβλητό αριθμό
# ορισμάτων με keywords
def myFun(**kwargs):
    for key, value in kwargs.items():
        print ("%s == %s" %(key, value))

# Driver code
myFun(first = 'Greeks', mid = 'for', last = 'Greeks')
```

Output

```
last == Greeks
mid == for
first == Greeks
```

```
# Παράδειγμα χρήσης ορισμάτων με **kwargs και 1 επιλέον
# όρισμα
def myFun(arg1, **kwargs):
    for key, value in kwargs.items():
        print ("%s == %s" %(key, value))
    for arg in argv:
        print("Next argument through *argv :", arg)
myFun(first = 'Greeks', mid = 'for', last = 'Greeks')
```


Παραδείγματα χρήσης συνάρτησης με *args, **kwargs

Παράδειγμα *args και **kwargs στην κλίση της συνάρτησης

```
def myFun(arg1, arg2, arg3):  
    print("arg1:", arg1)  
    print("arg2:", arg2)  
    print("arg3:", arg3)
```

```
# Now we can use *args or **kwargs to  
# pass arguments to this function :  
args = ("Greeks", "for", "Greeks")  
myFun(*args)
```

```
kwargs = {"arg1" : "Greeks", "arg2" : "for", "arg3" : "Greeks"}  
myFun(**kwargs)
```

Output

```
arg1: Greeks  
arg2: for  
arg3: Greeks  
arg1: Greeks  
arg2: for  
arg3: Greeks
```

Παράδειγμα *args και **kwargs στην κλίση της συνάρτησης

```
def myFun(*args,**kwargs):  
    print("args: ", args)  
    print("kwargs: ", kwargs)
```

```
# Now we can use both *args ,**kwargs  
# to pass arguments to this function :  
myFun('greeks','for','greeks',first="Greeks",mid="for",last="Greeks")
```

Τοπικές (**local**) και Γενικευμένες (**global**) μεταβλητές

Οι μεταβλητές σε ένα πρόγραμμα δεν είναι προσβάσιμες σε όλα τα σημεία του προγράμματος.

Αυτό εξαρτάται από το που έχουν οριστεί οι μεταβλητές

Ανάλογα με την έκταση εφαρμογής (**scope**) μιας μεταβλητής, στην Python ξεχωρίζουμε δύο είδη:

- **Τοπικές (local)** μεταβλητές
- **Γενικευμένες (global)** μεταβλητές

Μεταβλητές που ορίζονται στο σώμα μιας συνάρτησης έχουν τοπικό χαρακτήρα, ενώ αυτές που ορίζονται εκτός συναρτήσεων είναι γενικευμένες μεταβλητές.

Όσες μεταβλητές ορίζονται μέσα σε μια συνάρτηση είναι προσβάσιμες μόνο από εντολές της συνάρτησης ενώ μεταβλητές που ορίζονται εκτός της συνάρτησης είναι προσβάσιμες από όλες τις συναρτήσεις του προγράμματος.

Με την κλίση μιας συνάρτησης, οι μεταβλητές που ορίζονται μέσα στην συνάρτηση έρχονται σε ισχύ στο πεδίο εφαρμογής της συνάρτησης

Local και global μεταβλητές

Το παρακάτω παράδειγμα δείχνει τις περιπτώσεις των local και global μεταβλητών.

```
total = 0 # This is global variable.
glb     = 1 # This is global variable.

def sum( arg1, arg2 ):
    # Add both the parameters and return them
    total = arg1 + arg2 # total is local variable.
    test  = 2*glb        # test is local variable but glb global
    print("Inside the function local total : ", total, test)
    return total, test

a, b = sum( 10, 20 )
print("Outside the function global total : ", total, glb, a, b)
```

Το αποτέλεσμα του παραπάνω παραδείγματος θα είναι:

30, 2

0, 1, 30, 2

Namespace και scope

Έχουμε αναφέρει νωρίτερα ότι όταν κάνουμε `import` ένα `module` αυτόματα ορίζεται ένας χώρος με τα ονόματα όλων των συναρτήσεων/μεθόδων που σχετίζονται με το `module`

Ο χώρος αυτός ονομάζεται **namespace** του `module` και αποτελεί στην πραγματικότητα ένα `dictionary` με τα ονόματα των συναρτήσεων ως `keys` του `dictionary` και τα αντίστοιχα αντικείμενα αποτελούν τις τιμές των `keys`.

Μια εντολή `python` μπορεί να έχει πρόσβαση στο `global namespace` ή σε ένα `local namespace`.

Αν μια `local` και μια `global` μεταβλητή έχουν το ίδιο όνομα, τότε η `local` μεταβλητή επισκιάζει την `global`

Κάθε συνάρτηση σχετίζεται με το δικό της τοπικό `namespace` μεταβλητών

Για να αλλάξουμε την τιμή μιας `global` μεταβλητής μέσα σε μία συνάρτηση θα πρέπει να ορίσουμε την μεταβλητή ως `global` μέσα στη συνάρτηση, δίνοντας την εντολή:

global VarName

Η εντολή `global varname`

Η εντολή **global** `VarName` δηλώνει στην Python ότι `VarName` είναι global και παύει η Python να προσπαθεί να την βρει στο local namespace

```
Money = 2000
def AddMoney():
    Money = Money + 1
    print(Money)

AddMoney()
print(Money)
```

UnboundLocalError:
local variable 'money'
referenced before
assignment

fix

```
Money = 2000
def AddMoney():
    global Money
    Money = Money + 1
    print(Money)

AddMoney()
print(Money)
```

Οι συναρτήσεις **globals()** και **locals()** επιστρέφουν τα keys του dictionary του global namespace και ενός local namespace dictionary.

Αν οι συναρτήσεις **globals()** και **locals()** κληθούν μέσα από μια συνάρτηση, επιστρέφουν τα ονόματα των μεταβλητών που είναι global για τη συνάρτηση ή είναι local για τη συνάρτηση.

Οι συναρτήσεις αυτές επιστρέφουν dictionaries ως αντικείμενα με keys τα ονόματα όλων των global και local μεταβλητών και μπορούμε να τα εξαγάγουμε χρησιμοποιώντας τη συνάρτηση keys του dictionary.

Classes και **Αντικειμενοστραφής Προγραμματισμός**

Αντικειμενοστραφής Προγραμματισμός

Ο σχεδιασμός των προγραμμάτων γίνεται με τέτοιο τρόπο ώστε να δοθεί έμφαση στη λογική του αλγόριθμου και των συναρτήσεων που χρησιμοποιεί για την υποστήριξή του

- Μπορούμε να βελτιώσουμε τμήματα της επίλυσης ενός προβλήματος αν δομήσουμε ένα πρόγραμμα ώστε να χρησιμοποιεί συναρτήσεις για να εκτελέσουμε ή να υπολογίσουμε τμήματα του κώδικα που κάνουμε πολλές φορές

Ορίζουμε ή εισαγάγουμε δεδομένα που θέλουμε να αναλύσουμε ή να επεξεργαστούμε

- Σχεδιασμός συναρτήσεων για ανάγνωση ή αποθήκευση δεδομένων
- Σχεδιασμός δομών που θα βοηθήσουν να επεξεργαστούμε δεδομένα

Εν γένει είναι απαραίτητη η γνώση των διαδικασιών και των δομών που χρησιμοποιούνται ώστε να μπορέσουμε να χρησιμοποιήσουμε κάποιο πρόγραμμα ή να το τροποποιήσουμε

Στο τέλος θα μπορέσουμε να έχουμε κάποιο αποτέλεσμα: ...

Παράδειγμα ορισμού διαδικασίας

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def falling_ball(x0,v0,g,t):
    return x0+v0*t + 0.5*g*t*t
```

Ορισμός συνάρτησης/ων που χρησιμοποιούμε συχνά

```
def main():
    # Group metavlitwn
    x0 = 100.
    v0 = 0
    g = 9.8
    t = np.arange(101.)
    # Prosdiiorismos tou pinaka eksodou
    x = np.zeros(101)
    # Ypologismos
    for i in range(101):
        x[i] = falling_ball(x0,v0,g,tt[i])
    # Epeksergasia dedomenwn
    pl.plot(t,x)
```

Αρχικοποίηση μεταβλητών που χρησιμοποιούμε στους υπολογισμούς

Δημιουργία κατάλληλης δομής για να κρατήσουμε τα δεδομένα προσέχοντας να έχει το σωστό μέγεθος

Εκτέλεση των υπολογισμών . Πρέπει να είμαστε προσεκτικοί ώστε στο τέλος να μην έχουμε παραλείψει κάτι

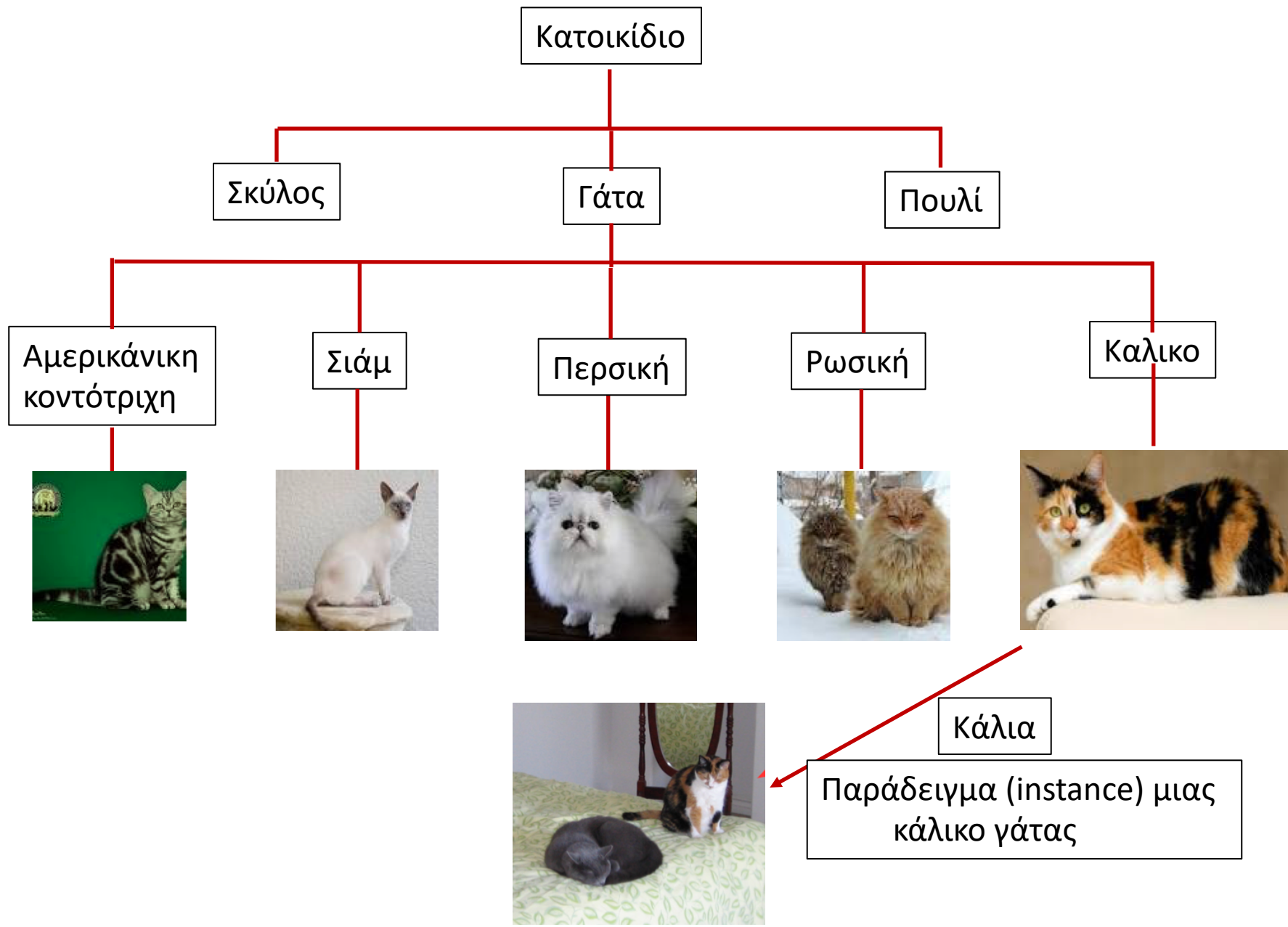
Να εκτελέσουμε κάτι για να εξετάσουμε από τα δεδομένα

```
main()
```


Αντικειμενοστραφής προγραμματισμός

- Έμφαση δίνεται στο να γραφούν «αντικείμενα»
- Τα αντικείμενα περιέχουν δεδομένα τα οποία διατηρούνται σε μια δομή
- Τα αντικείμενα περιέχουν μεθόδους/συναρτήσεις που ενεργούν στα δεδομένα
- Τα αντικείμενα έχουν συγκεκριμένους τρόπους και μεθόδους που αλληλεπιδρούν με άλλα αντικείμενα και χρήστες
- Επιμέρους αντικείμενα μπορεί να ανήκουν στην ίδια κατηγορία (**class**) αντικειμένων. Ίδιο είδος αντικειμένου απλά διαφορετικά δεδομένα και ιδιότητες

Class, αντικείμενο, στιγμιότυπο



Τι γνωρίζουμε για τις γάτες...

- ❑ Οι γάτες έχουν δεδομένα που σχετίζονται με αυτές (attributes)
 - Όνομα
 - Ράτσα
 - DNA
- ❑ Οι γάτες κάνουν πράγματα (έχουν ιδιότητες/μεθόδους)
 - Τρώνε
 - Παίζουν
 - Κυνηγούν
- ❑ Εσωτερικά (ως αντικείμενα) οι γάτες είναι ιδιαίτερα σύνθετες ωστόσο μπορούμε να τις χρησιμοποιήσουμε. Δεν χρειάζεται να ξέρει κάποιος το DNA μιας γάτας για να μπορέσει να παίξει μαζί της

Αντικειμενοστραφής προγραμματισμός

- ❑ Τα αντικείμενα είναι εύκολα να χρησιμοποιηθούν, να προεκταθούν και να επαναχρησιμοποιηθούν
- ❑ Τα αντικείμενα αποκρυσταλλώνουν τα δεδομένα τους:
 - Ο τρόπος με τον οποίο δουλεύουν στο εσωτερικό τους παραμένει κρυφό ώστε να αποφεύγονται συγχίσεις και μόνο προσφέρουν πρόσβαση στα δεδομένα τους με πολύ καλά ορισμένες διεπαφές
 - Ο χρήστης δεν χρειάζεται να γνωρίζει με λεπτομέρεια το τρόπο επεξεργασίας των διάφορων δομών που απαιτούνται για την πρόσβαση στα δεδομένα ενός αντικειμένου
- ❑ Τα αντικείμενα μπορούν να επεκταθούν ώστε να δημιουργήσουν νέα αντικείμενα τα οποία κληρονομούν τις ιδιότητες του προϋπάρχοντος αντικειμένου.
 - Δεν χρειάζεται να γραφούν πολλές γραμμές κώδικα για να ορίσουμε κάτι το οποίο είναι αρκετά παρόμοιο με κάτι το οποίο υπάρχει

Κριτική για αντικειμενοστραφή προγραμματισμό

❑ Απόδοση

- Απαιτείται περισσότερος χρόνος για τον σχεδιασμό ενός τέτοιου προγράμματος.
- Χρειάζεται να γραφεί περισσότερος κώδικας ώστε το πρόγραμμα να αναφέρεται σε περισσότερα πράγματα.

❑ Δεν είναι όλα αντικείμενα. Επομένως η ερώτηση είναι γιατί θα πρέπει να χρησιμοποιήσουμε αυτή την προσέγγιση

- Οι αλγόριθμοι και ο υπολογισμός είναι εξίσου σημαντικοί.
- Γιατί θα πρέπει με τον τρόπο αυτό να δώσουμε περισσότερη έμφαση στα «ουσιαστικά» από ότι στα «ρήματα»
- Πως λαμβάνεται υπ' όψην ο χρόνος
- Πως λαμβάνεται υπόψην παράλληλη επεξεργασία (parallel processing)

Χαρακτηριστικά λογισμικών αντικειμένων

❑ Attributes ή χαρακτηριστικά (δεδομένα)

- **Αφηρημένες έννοιες**: Πως θα φαίνεται το αντικείμενο εξωτερικά; Ποιες είναι οι βασικές λεπτομέρειες του αντικειμένου που θα επιτρέψουν να είναι εύχρηστο
- **Αποκρυστάλλωση**: Πως χειρίζονται τα δεδομένα στο εσωτερικό του αντικειμένου ώστε να επιτευχθεί η επιθυμητή συμπεριφορά; Ποιες είναι οι βασικές λεπτομέρειες που δεν χρειάζονται για να χρησιμοποιηθεί ένα αντικείμενο.

❑ Μέθοδοι – συναρτήσεις που ανήκουν στα αντικείμενα

❑ «Απόκρυψη» δεδομένων

- **Δημόσια (public)**: διαθέσιμα να χρησιμοποιηθούν από εξωτερικές συναρτήσεις, εκτός του αντικειμένου. Προσδιορίζουν τι κάνει το αντικείμενο
- **Ιδιωτικά (private) ;ή προστατευόμενα**: διαθέσιμα μόνο εσωτερικά στο αντικείμενο και δεν είναι προσβάσιμες εξωτερικά. Χρησιμοποιούνται για να επιτευχθεί η επιθυμητή συμπεριφορά

❑ «Κληρονομικότητα» (**inheritance**) – Δυνατότητα να επεκταθεί ένα προϋπάρχον αντικείμενο και να δημιουργηθούν νέα αντικείμενα

Αντικειμενοστραφής προγραμματισμός και Python

- ❑ Η Python είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού από σχεδιασμό
 - Τα πάντα στην γλώσσα Python είναι μια class
 - Η Python έχει την δυνατότητα να φτιάχνει νέες κλάσεις επιτρέποντας έτσι χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού να εφαρμοστούν
- ❑ Χρειάζεται να μάθουμε να γράφουμε και να χρησιμοποιούμε κλάσεις Python
 - Θα συζητήσουμε ένα παράδειγμα κλάσης, MyVector, η οποία θα χειριστεί διανύσματα και πράξεις μεταξύ διανυσμάτων

Εφαρμογή μιας κλάσης στην Python

```
import math
```

```
class MyVector:
```

```
    def __init__(self, x, y, z):
```

```
        #constructor
```

```
        self.x = x
```

```
        self.y = y
```

```
        self.z = z
```

```
    def __str__(self):
```

```
        return 'MyVector(%f,%f,%f)'%(self.x,self.y,self.z)
```

```
    def __add__(self, other):
```

```
        return MyVector(self.x+other.x, self.y+other.y, self.z+other.z)
```

```
    def norm(self):
```

```
        return math.sqrt(self.x**2+self.y**2+self.z**2)
```

`__init__` είναι η συνάρτηση που χτίζει την Class και ονομάζεται constructor.

Δημιουργεί στιγμιότυπο της class MyVector

x,y,z είναι τα δεδομένα του διανύσματος

`self` είναι η μεταβλητή αναφοράς του στιγμιότυπου της class MyVector

`str` είναι η μέθοδος που είναι διαθέσιμη σε όλες τις κλάσεις της Python ώστε να κάνει τη string διαθέσιμη στη συνάρτηση print

`add` είναι η μέθοδος που περιγράφει πως να διαχειριστεί τον τελεστή +

`norm` είναι η δική μας μέθοδος για τη συγκεκριμένη class

Χρήση της Class MyVector

```
>>> from MyVector import MyVector
```

```
>>> a = MyVector(3.,1.,0.)
```

```
>>> print a  
MyVector (3.000000,1.000000,0.000000)
```

```
>>> b = MyVector(2.,2.,2.)
```

```
>>> print(a+b)  
MyVector(5.000000,3.000000,2.000000)
```

```
>>> print(a.norm())  
3.16227766017
```

Εισαγωγή της class MyVector από το αρχείο MyVector.py

Δημιουργία ενός στιγμιότυπου της class MyVector

Εκτύπωση του a χρησιμοποιώντας την εντολή print που χρησιμοποιεί τη μέθοδο `__str__` του MyVector

Δημιουργία ενός άλλου διανύσματος

Πρόσθεση των διανυσμάτων $a+b$ και εκτύπωση του αποτελέσματος

Εκτύπωση του μέτρου του διανύσματος a

Προέκταση της class MyVector

```
import math
```

```
class MyPolarVector(MyVector):  
    def __init__(self, r, t, p):  
        #constructor  
        MyVector.__init__(self,  
                            r*math.cos(t)*math.cos(p),  
                            r*math.cos(t)*math.sin(p),  
                            r*math.sin(t))  
  
    def r(self):  
        return self.norm()  
  
    def phi(self):  
        return math.atan2(self.y, self.x) ]  
  
    def theta(self):  
        return math.asin(self.z / self.norm())
```

Νέα class MyPolarVector που κληρονομεί όλες τις συναρτήσεις της MyVector. Μπορούμε να προσθέσουμε νέες μεθόδους

Αρχικά ο constructor πρέπει να δημιουργήσει ένα στιγμιότυπο της αρχικής class MyVector

Δημιουργία άλλων μεθόδων

Χρήση του MyPolarVector

<pre>>>> from MyVector import MyVector, MyPolarVector</pre>	<pre>import MyVector και MyPolarVector από το αρχείο MyVector.py</pre>
<pre>>>> a = MyVector(3.,1.,0.) >>> b = MyPolarVector(3.,1.,0.)</pre>	<p>Δημιουργία ενός στιγμιότυπου και των δύο MyVector και MyPolarVector</p>
<pre>>>> print(b) MyVector(1.620907,0.000000,2.524413)</pre>	<p>Εκτύπωση του b MyPolarVector χρησιμοποιεί την <code>__str__</code> του MyVector</p>
<pre>>>> print(a+b) MyVector(4.620907,1.000000,2.524413)</pre>	<p>Η πρόσθεση χρησιμοποιεί τη μέθοδο <code>add</code> του MyVector</p>
<pre>>>> print(b.r()) 3 >>> print(b.theta()) 1.0 >>> print(b.phi()) 0.0</pre>	<p>Χρήση των νέων μεθόδων από την κλάση MyPolarVector</p>

Χαρακτηριστικά της class `MyVector`

Χρήση των διανυσμάτων σαν μια οντότητα. Αυτό απλουστεύει τα πράγματα

Αφηρημένες έννοιες: (abstraction)

Ο τρόπος που ενεργούμε είναι ο τρόπος που εφαρμόζεται στις πράξεις με διανύσματα. Δεν μας απασχολεί ο τρόπος με τον οποίο γίνονται αυτές οι πράξεις.

Αποκρυστάλλωση ιδιοτήτων:

Μετά τον ορισμό δεν χρειάζεται να ανησυχούμε για τις συνιστώσες και τον τρόπο που περιγράφονται στην class. Για παράδειγμα θα μπορούσαμε να είχαμε χρησιμοποιήσει NumPy πίνακες και τελεστές (πράξεις) για να κάνουμε τις πράξεις με τα διανύσματα και ο χρήστης δεν θα καταλάβαινε τη διαφορά