Python – Εννοιες Αντικειμενοστρεφους Προγραμματισμου και Εφαρμογη στην Πραξη

Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

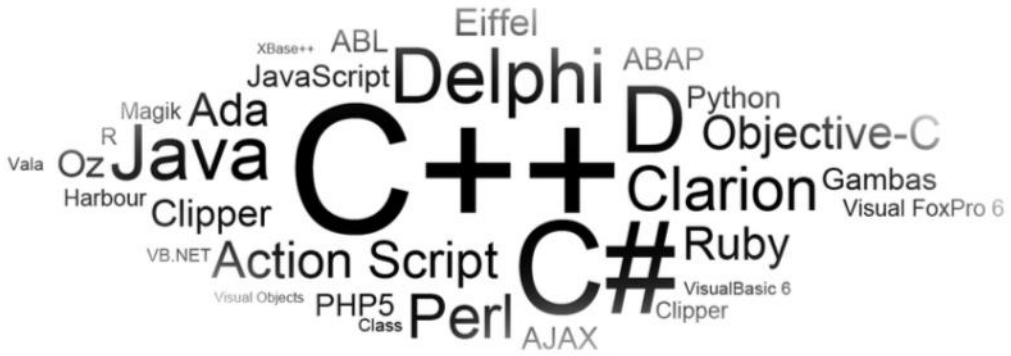
Γιάννης Γαροφαλάκης - Σπύρος Σιούτας - Γιάννης Τζήμας

Το πρώτο πράγμα που πρέπει πάντα να κάνετε είναι να ρωτάτε το **γιατί;**

- Σε αυτό το μάθημα θα :
  - Δούμε από πιο κοντά έννοιες αντικειμενοστρεφούς προγραμματισμού στην Python και θα
  - Αξιοποιήσουμε αυτά που θα μάθουμε σε πραγματικά παραδείγματα.

Αντικειμενοστρεφής Προγραμματισμός (1/3)

- Ο Αντικειμενοστρεφής προγραμματισμός (Object Oriented Programming - OOP) είναι ένας τρόπος προγραμματισμού που υποστηρίζεται από πάρα πολλές και διαδεδομένες γλώσσες προγραμματισμού όπως τις:
  - Python
  - C#
  - C++
  - Java και πολλές ακόμα.

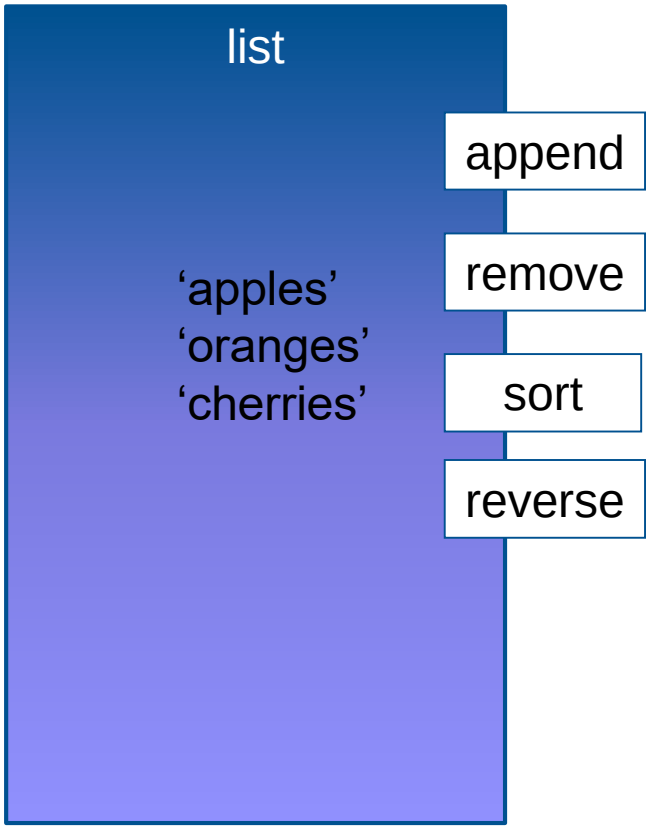


Αντικειμενοστρεφής Προγραμματισμός (2/3)

- Μέχρι τώρα είδαμε πώς να χρησιμοποιούμε αφαιρέσεις και ο αντικειμενοστρεφής προγραμματισμός έχει σαν πυρήνα τις αφαιρέσεις, γιατί τα αντικείμενα είναι αφαιρέσεις – οργανώνουν τη λογική και τα δεδομένα μας μειώνοντας την πολυπλοκότητα.

- Αντί να έχουμε ένα γιγαντιαίο πρόγραμμα με κώδικα και μεταβλητές παντού, μπορούμε να έχουμε αντικείμενα που ομαδοποιούν των κώδικα και τις μεταβλητές μας.

- Έχουμε ήδη χρησιμοποιήσει αντικείμενα όπως η λίστα της Python.
  - Η λίστα είναι ένα αντικείμενο που είναι δομή δεδομένων – ένα δοχείο δεδομένων. Όπως θυμάστε από το προηγούμενο μάθημα, μπορούμε σε μία λίστα να αποθηκεύσουμε τα αντικείμενα ενός καλαθιού αγορών.
  - Επίσης, όμως εκθέτει μεθόδους για να διαχειριστούμε αυτά τα δεδομένα.



Αντικειμενοστρεφής Προγραμματισμός (3/3)

- Ένας λοιπόν από τους στόχους του OOP είναι να πάρεις ότι είναι απαραίτητο σε σχέση με μία εφαρμογή και να βρεις τρόπο να δημιουργήσεις αντικείμενα που να ομαδοποιήσουν τον κώδικα και τα δεδομένα που χρειάζεσαι. Αυτό μειώνει την πολυπλοκότητα.

- Για να φτιάξουμε όμως το πρόγραμμά μας, εκτός από τα έτοιμα αντικείμενα που μας δίνει η Python χρειαζόμαστε και αντικείμενα που θα ορίσουμε εμείς και αναπαριστούν τις ανάγκες της εφαρμογής μας.

- Τα αντικείμενα που θα δημιουργήσουμε μπορεί να αναπαριστούν αντικείμενα του πραγματικού κόσμου, αλλά αυτό δεν είναι και απαραίτητο.

Ας φτιάξουμε λοιπόν ένα δικό μας αντικείμενο…

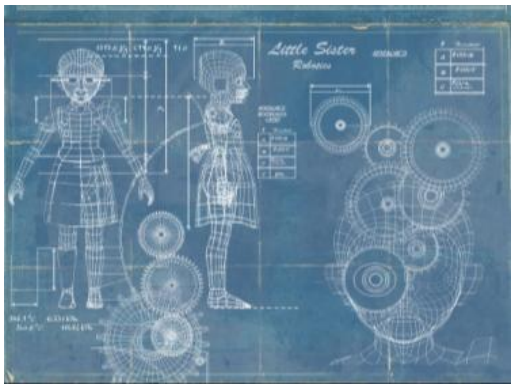
- Όταν δημιουργούμε ένα αντικείμενο θα πρέπει να υπάρχει ένα σχεδιάγραμμα – blueprint για αυτό, ορισμένο κάπου.
- Η κλάση είναι αυτό το σχεδιάγραμμα.
- Ας φτιάξουμε ένα αντικείμενο **`Person`**.
- Μία μέθοδος είναι μία συνάρτηση που έχει συσχετιστεί με ένα αντικείμενο.

Μία σύμβαση που χρησιμοποιούμε στην Python είναι το πρώτο γράμμα της κλάσης να είναι κεφαλαίο.

```
class Person:

    def say_hello(self):
        print("Hello!")


p1 = Person()
p1.say_hello()
```

```
>>> ================================ RESTART ================
>>>
Hello!
>>>
```

Δεν μπορούμε να χρησιμοποιήσουμε τον ορισμό της κλάσης σαν αντικείμενο, αλλά μπορούμε να δημιουργήσουμε αντικείμενα από την κλάση. Αυτή η διαδικασία λέγεται **Instantiation**.
Δεν μπορώ να γράψω **`Person.say_hello()`** .



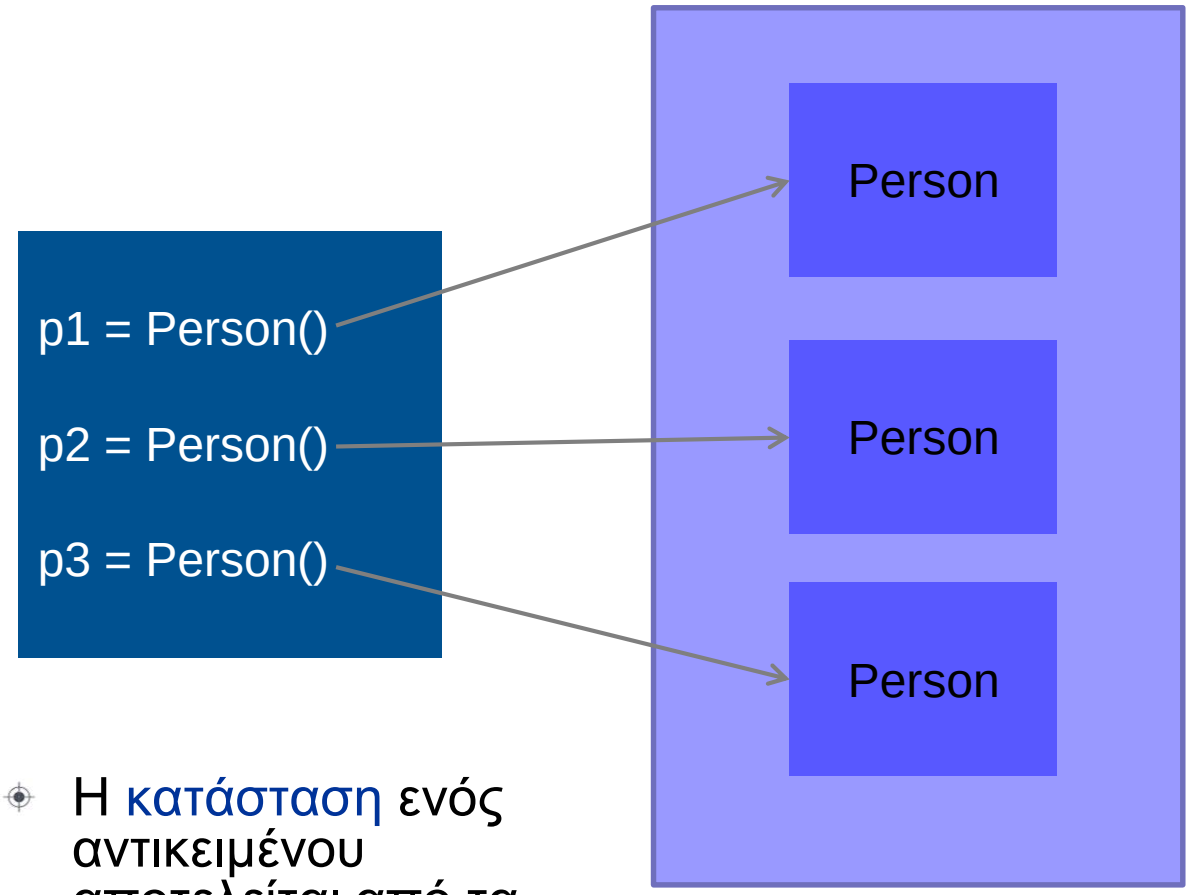
Instantiation (1/2)

- Μπορούμε να κάνουμε instantiate πολλαπλά αντικείμενα από τον ίδιο ορισμό κλάσης και αυτό είναι πολύ χρήσιμο.

- Για να γράψουμε ένα μεγάλο πρόγραμμα σίγουρα χρειαζόμαστε πολλαπλά αντικείμενα.

- Για την περίπτωσή μας τα αντικείμενα Person θα αναπαριστούν διαφορετικούς ανθρώπους.

- Κάθε ένα από αυτά τα αντικείμενα θα δημιουργείται διακριτά στη μνήμη και θα κρατάει διαφορετικά τμήματα δεδομένων. Αυτό το ονομάζουμε **κατάσταση** / **state** του αντικειμένου.



- Η κατάσταση ενός αντικειμένου αποτελείται από τα δεδομένα που κρατάει.

Instantiation (2/2)

- Προς το παρόν δεν έχουμε δεδομένα σε κάθε αντικείμενο και ας δούμε πως γίνεται αυτό.

- Όλες οι γλώσσες που επιτρέπουν τη δημιουργία αντικειμένων από κλάσεις, έχουν κάποια ειδική μέθοδο για την αρχικοποίηση των αντικειμένων με συγκεκριμένα κομμάτια δεδομένων.

- Κάποιες γλώσσες αποκαλούν αυτές τις μεθόδους **constructors / κατασκευαστές**, γιατί χρησιμοποιούνται για την κατασκευή ενός αντικειμένου.

- Στην Python αυτή η μέθοδος είναι η **`init`** (από το initialize).

```
class Person:

    def __init__(self):
        self.name = "John"

    def say_hello(self):
        print("Hello!", self.name)


p1 = Person()
p1.say_hello()
```

```
>>> =============
>>>
Hello! John
>>>
```

Η μέθοδος init

- Αυτός ο τρόπος γραφής είναι μία σύμβαση για μερικές από τις ειδικές μεθόδους της Python, με τις οποίες δεν αλληλεπιδρούμε. Δε θα γράφαμε ποτέ **`p1.__init__`**

- Αυτό το κάνει η Python για εμάς όταν κατασκευάζουμε ένα αντικείμενο.

- Όταν κάνουμε instantiate ένα αντικείμενο το πρώτο πράγμα που κάνει η Python είναι να εκτελέσει την init μέθοδο για εμάς. Για αυτό το λόγο είναι ο initializer (constructor). Μου επιτρέπει φτιάξω κάποια πράγματα εντός του αντικειμένου πριν εκτελεστεί ο υπόλοιπος κώδικας.

- Η παράμετρος που περνάμε στην **`__init__`** δε χρειάζεται να είναι η **`self`**, αλλά αυτό είναι μία ακόμα σύμβαση.

```
class Person:

    def __init__(self):
        self.name = "John"

    def say_hello(self):
        print("Hello!", self.name)


p1 = Person()
p1.say_hello()
```

Δεν χρειάζεται να περάσουμε κάποια παράμετρο όταν καλούμε κάποια μέθοδο του αντικειμένου και αυτό γιατί η ίδια η Python θα περάσει μία παράμετρο η οποία ουσιαστικά θα είναι μία αναφορά στο αντικείμενο όπου αυτή η μέθοδος καλείται.

Attributes - Κατηγορήματα

- Τα **attributes** / **κατηγορήματα** είναι στην πραγματικότητα τα τμήματα των δεδομένων τα οποία το αντικείμενο εκθέτει. Σε άλλες γλώσσες ονομάζονται properties.

- Και αν θέλουμε να το κάνουμε λίγο ποιο ενδιαφέρον, ώστε όλα τα αντικείμενα να μην έχουν το ίδιο **name**.

```
class Person:

    def __init__(self):
        self.name = "John"

    def say_hello(self):
        print("Hello!", self.name)


p1 = Person()
p1.say_hello()
```

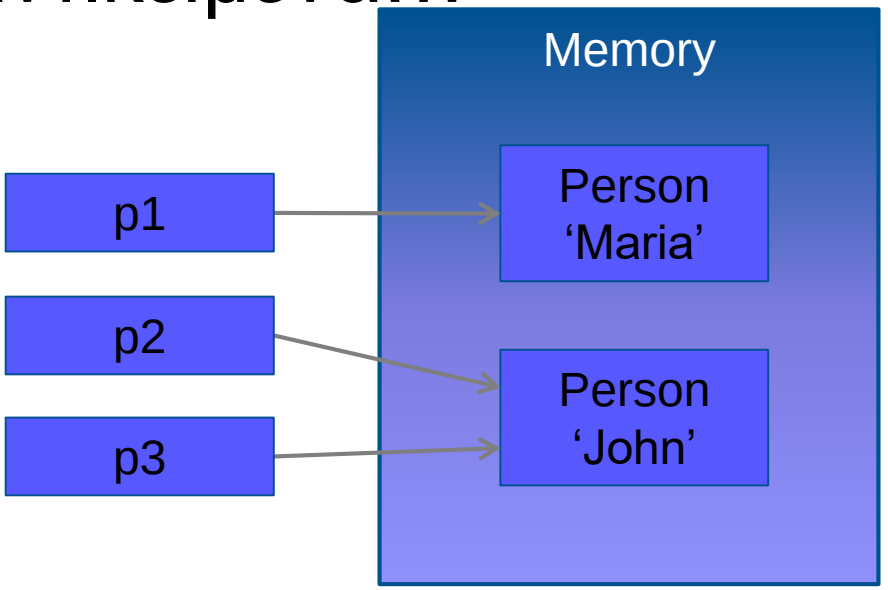
```
class Person:

    def __init__(self):
        self.name = "John"

    def say_hello(self):
        print("Hello, ", self.name)


p1 = Person()
p1.name = "Maria"
p1.say_hello()

p2 = Person()
p2.say_hello()
```

```
>>>
Hello,  Maria
Hello,  John
>>>
```

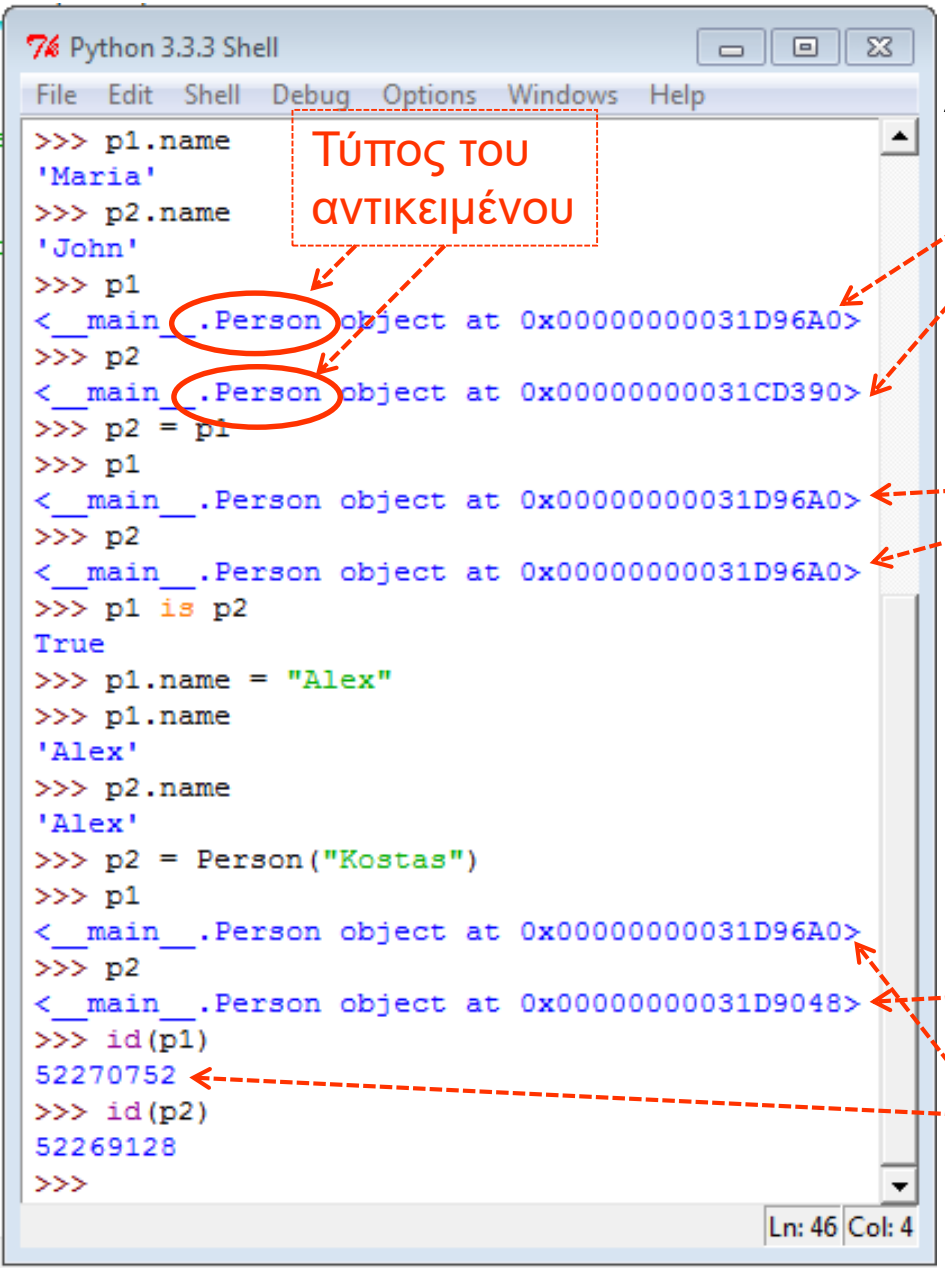
```
class Person:

    def __init__(self, name):
        self.name = name

    def say_hello(self):
        print("Hello, ", self.name)


p1 = Person("Maria")
p1.say_hello()

p2 = Person("John")
p2.say_hello()
```

Και ας δούμε τώρα τη συσχέτιση των μεταβλητών με τα αντικείμενα…

- Όταν δημιουργούμε ένα αντικείμενο αυτό καταλαμβάνει κάποιο χώρο στη μνήμη του υπολογιστή.
- Οι μεταβλητές δεν περιέχουν τα αντικείμενα. Τις χρησιμοποιούμε για να έχουμε πρόσβαση σε αυτά.
- Τις μεταβλητές μπορούμε να τις σκεφτόμαστε σα **δείκτες** ή **αναφορές**.
- Περιέχουν τη θέση που μπορούμε να βρούμε ένα αντικείμενο.
- Μία μεταβλητή μπορεί να δείχνει ένα αντικείμενο κάθε φορά, αλλά **δύο διαφορετικές μεταβλητές μπορούν να δείχνουν στο ίδιο αντικείμενο**.



Αντικείμενα



Τύπος του αντικειμένου

Διεύθυνση μνήμης (Δεκαεξαδικό)

Δείχνουν πλέον στο ίδιο αντικείμενο, άρα και στην ίδια θέση μνήμης.

Πλέον δείχνει σε άλλο αντικείμενο.

Είναι η ίδια τιμή σε δεκαδικό.

Και στην πράξη…

```
class Person:

    def __init__(self, name):
        self.name = name

    def say_hello(self):
        print("Hello, ", self.name)


p1 = Person("Maria")
p2 = p1

p2.name = "Kostas"
p1.say_hello()
```

```
>>>
Hello,  Kostas
>>>
```

```
class Person:

    def __init__(self, name):
        self.name = name

    def say_hello(self):
        print(id(self))
        print("Hello, ", self.name)


p1 = Person("Maria")
p2 = Person("John")
print(id(p1))
print(id(p2))
p1.say_hello()
```

```
class Person:

    def __init__(self, name):
        self.name = name

    def say_hello(self):
        print(id(self))
        print("Hello, ", self.name)


p1 = Person("Maria")
p2 = p1
p2.name = "Kostas"
print(id(p1))
print(id(p2))
p1.say_hello()
```

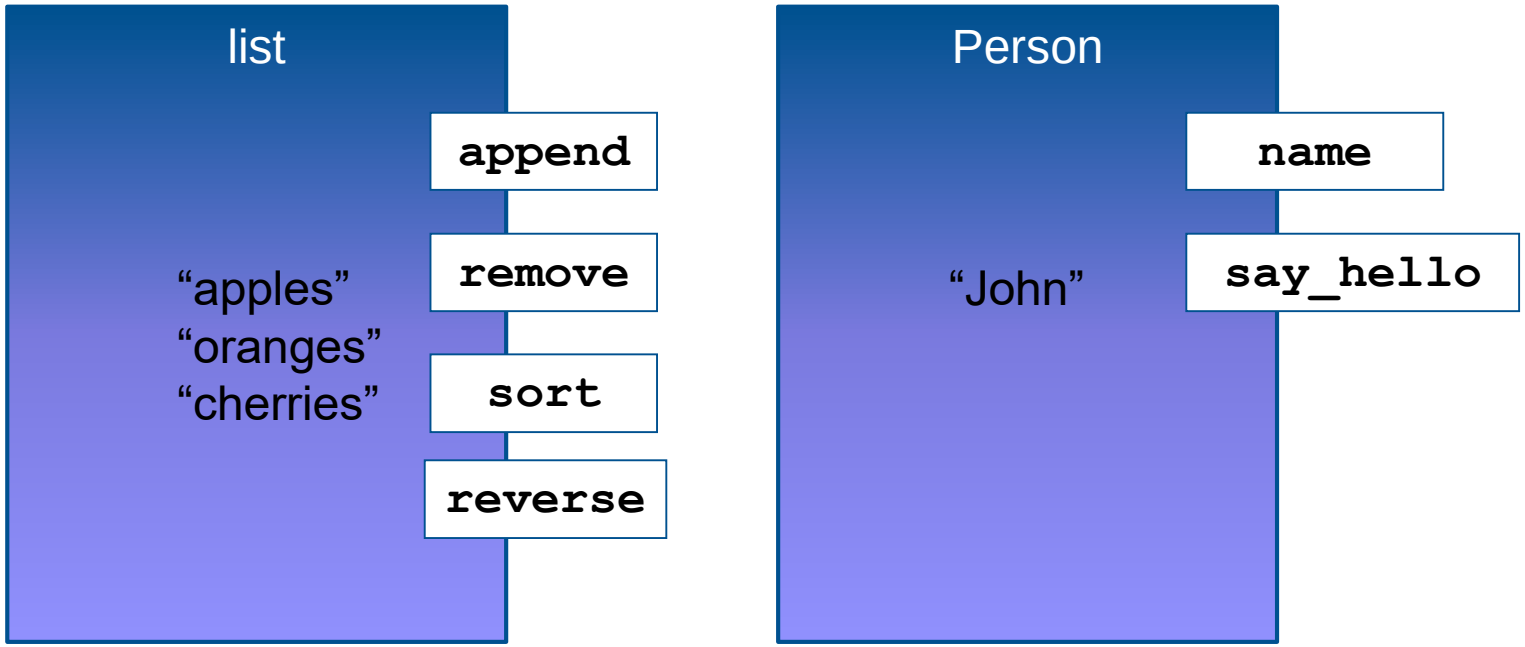
```
>>>
53012128
53012128
53012128
Hello,  Kostas
>>>
```

```
>>>
54038656
54063792
54038656
Hello,  Maria
>>>
```

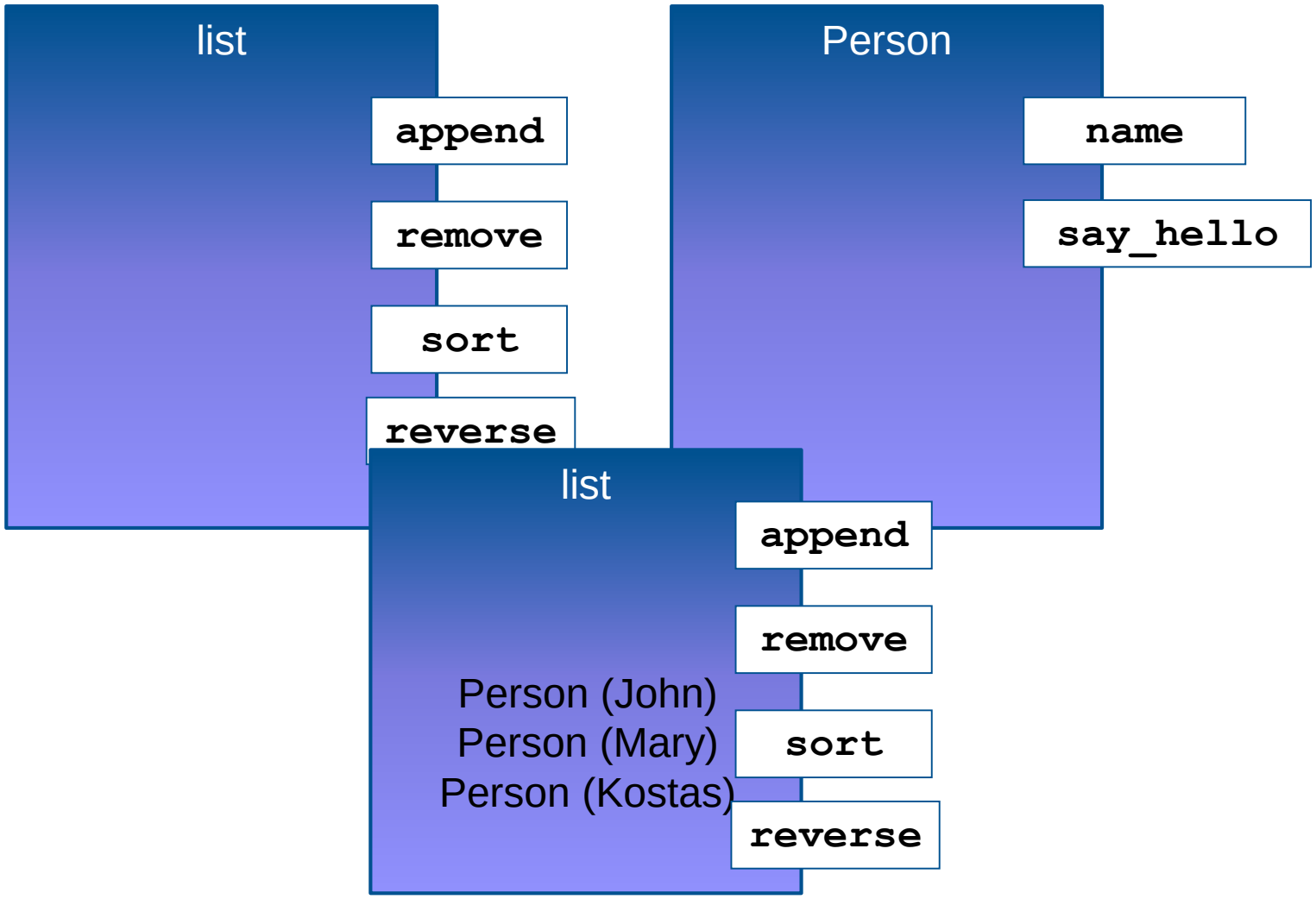
Ενθυλάκωση – Encapsulation

- Οι αντικειμενοστρεφείς τεχνικές είναι δημοφιλείς τα τελευταία 20 χρόνια γιατί μας επιτρέπουν να δημιουργήσουμε αφαιρέσεις (γενικεύουν και απλοποιούν τον κώδικά μας μειώνοντας την πολυπλοκότητά του).

- Ένας τρόπος για να γίνει αυτό είναι η ενθυλάκωση. Υπάρχουν δύο τρόποι για να σκέφτεστε την ενθυλάκωση:
  - Απόκρυψη λεπτομερειών (ξέρετε πως ακριβώς λειτουργεί ένα list ή πως ακριβώς λειτουργεί ένα αυτοκίνητο πριν το οδηγήσετε;)
  - Ομαδοποίηση λεπτομερειών σε ένα συνεκτικό μοντέλο: Κλάσεις (Ενσωματωμένα δεδομένα και συμπεριφορά).



Σύνθεση – Composition



- Το πώς συνθέτουμε όλα μαζί τα διαφορετικά αντικείμενα για να κάνουμε κάτι μεγαλύτερο – μία πιο στοχευμένη αφαίρεση.

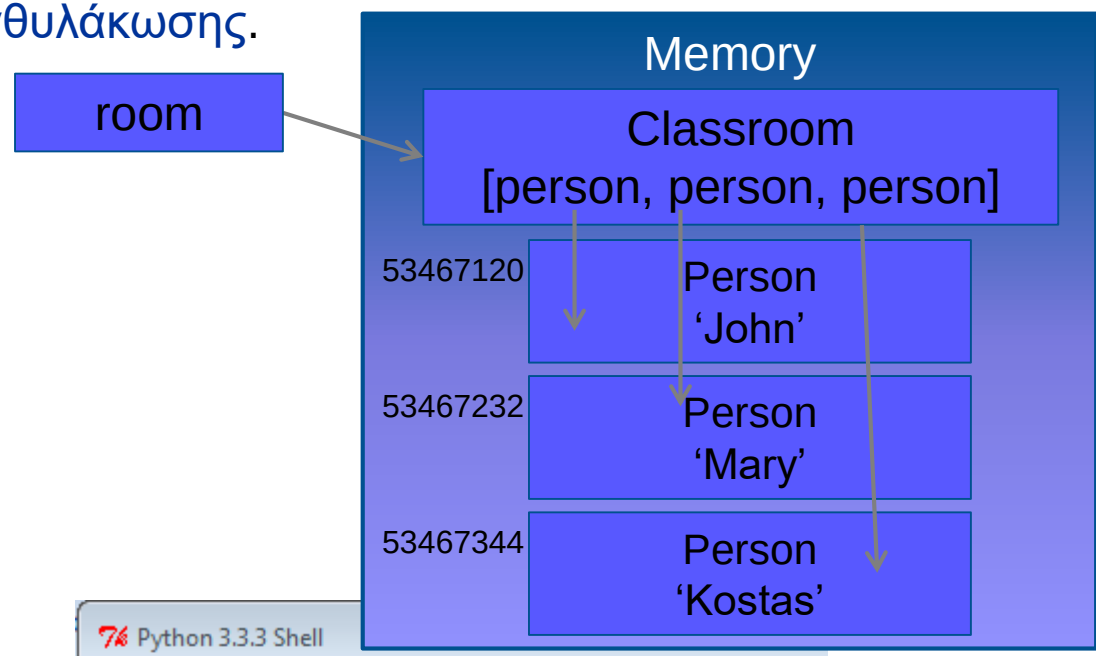
Ας συνθέσουμε…

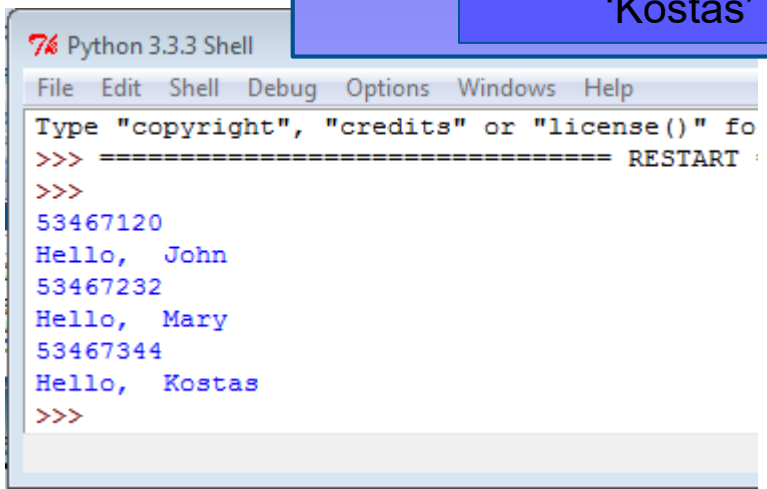
- Γυρνώντας στο πρόγραμμά μας σκεφτείτε την περίπτωση που διαχειριζόμαστε μία τάξη.
- Μία τάξη μπορεί να περιέχει πολλά άτομα.
- Θα πρέπει να φτιάξουμε μία αφαίρεση για να διαχειριζόμαστε πολλαπλά αντικείμενα τύπου Person και να τα ομαδοποιούμε σε μία τάξη.

…στην πράξη…

```
class Classroom:

    def __init__(self):
        self._people = []

    def add_person(self, person):
        self._people.append(person)

    def remove_person(self, person):
        self._people.remove(person)

    def greet(self):
        for person in self._people:
            person.say_hello()
class Person:

    def __init__(self, name):
        self.name = name

    def say_hello(self):
        print(id(self))
        print("Hello, ", self.name)

room = Classroom()
room.add_person(Person("John"))
room.add_person(Person("Mary"))
room.add_person(Person("Kostas"))

room.greet()
```

Σύμβαση: Όταν ονοματίζετε ένα attribute με _ σαν πρώτο γράμμα, τότε θα χρησιμοποιείται μόνο εντός της κλάσης. Είναι μία μορφή ενθυλάκωσης.





Τι είδαμε μέχρι τώρα;

- Είδαμε τις κλάσεις της Python οι οποίες έχουν attributes και methods.
- Μας βοηθάνε να δημιουργήσουμε αφαιρέσεις.
- Είδαμε επίσης τις έννοιες της ενθυλάκωσης και της σύνθεσης.

```
def get_order():
    print("[command] [item] (command is a to add, d to delete, q to quit)")
    line = input()

    command = line[:1]
    item = line[2:]

    return command, item

def add_to_cart(item, cart):
    if not item in cart:
        cart[item] = 0
    cart[item] += 1

def delete_from_cart(item, cart):
    if item in cart:
        cart[item] -= 1

def process_order(order, cart):
    command, item = order

    if command == "a":
        add_to_cart(item, cart)
    elif command == "d" and item in cart:
        delete_from_cart(item, cart)
    elif command == "q":
        return False

    return True

def go_shopping():
    cart = dict()

    while True:
        order = get_order()
        if not process_order(order, cart):
            break


    print(cart)
    print("Finished!")


go_shopping()
```

Στην Πράξη

- Ας εφαρμόσουμε στην πράξη όσα μάθαμε στο καλάθι αγορών.
- Μέχρι το σημείο που είχαμε φτάσει το πρόγραμμά μας χρησιμοποιούσαμε μόνο συναρτήσεις. Και με αυτές δημιουργούμε αφαιρέσεις.
- Κοιτώντας τον κώδικα που γράψαμε βλέπουμε ότι όλα κινούνται γύρω από δύο έννοιες.
  - Την παραγγελία – order
  - Το καλάθι αγορών – cart

Τι πρέπει να σκεφτόμαστε όμως για να επιλέξουμε τις κλάσεις μας;

- Αρχή #1: Να δίνετε στις κλάσεις σας μία μοναδική ευθύνη. Για το καλάθι μας η ευθύνη αυτή ίσως είναι απλά να αποθηκεύει τα αντικείμενα που έχει επιλέξει ο χρήστης. Οι κλάσεις σας πρέπει να είναι μικρές και στοχευμένες.

```
class Cart:

    def __init__(self):
        self._contents = dict()

class Order:
```

- Αρχή #2: Να σχεδιάζετε τις κλάσεις σας από έξω. Γράψτε πρώτα τον κώδικα που θα χρησιμοποιήσει την κλάση πριν από τον κώδικα της κλάσης.

```
cart = Cart()
order = Order()
order.get_input()

while not order.quit:

    order = Order()
    order.get_input()
```

Και αφού τα σκεφτήκαμε όλα ας φτιάξουμε τις κλάσεις μας

```
class Cart:

    def __init__(self):
        self._contents = dict()

class Order:

    def __init__(self):
        self.quit = False

    def get_input(self):
        print("[command] [item] (command is a to add, d to delete, q to quit)")
        line = input()

        command = line[:1]
        item = line[2:]

        if command == "q":
            self.quit = True


cart = Cart()
order = Order()
order.get_input()

while not order.quit:

    order = Order()
    order.get_input()
```

- Φτιάχνουμε αρχικά τις κλάσεις μας.
- Δανειζόμαστε λίγο από τον κώδικα που είχαμε γράψει και μπορούμε να ξεκινήσουμε

```
>>> ================================ RESTART ================================
>>>
[command] [item] (command is a to add, d to delete, q to quit)
q
>>>
```

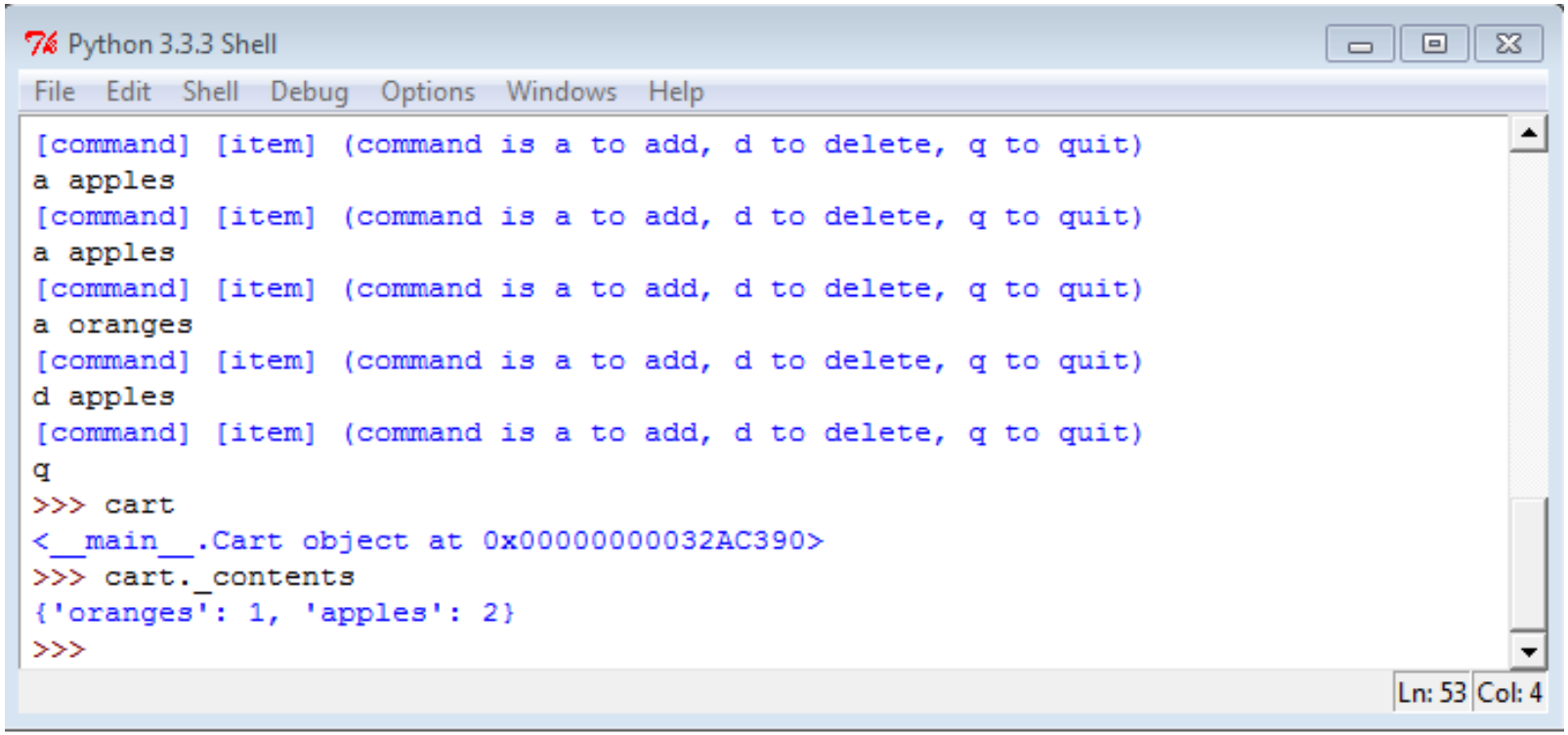
```
class Cart:

    def __init__(self):
        self._contents = dict()

    def process(self, order):
        if order.add:
            if not order.item in self._contents:
                self._contents[order.item] = 0
            self._contents[order.item] += 1
        elif order.delete:
            if order.item in self._contents:
                self._contents[order.item] -= 1
                if self._contents[order.item] <=0:
                    del self._contents[order.item]

class Order:

    def __init__(self):
        self.quit = False
        self.add = False
        self.delete = False
        self.item = None

    def get_input(self):
        print("[command] [item] (command is a to add, d to delete, q to quit)")
        line = input()

        command = line[:1]
        self.item = line[2:]

        if command == "a":
            self.add = True
        elif command == "d":
            self.delete == True
        elif command == "q":
            self.quit = True
```

Κυρίως πρόγραμμα

```
cart = Cart()
order = Order()
order.get_input()

while not order.quit:
    cart.process(order)
    order = Order()
    order.get_input()
```

Αντί να κάνουμε το command attribute και η επεξεργασία να γίνει στο process αφήνουμε αυτή τη διαδικασία στο Order.

Αυτό που έχει το καλάθι μας μέχρι στιγμής



Python 3.3.3 Shell

File Edit Shell Debug Options Windows Help

```
>>> ================================ RESTART ================================
>>>
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
d apples
[command] [item] (command is a to add, d to delete, q to quit)
q
>>> cart
<class '__main__.Cart'> {'_contents': {'apples': 2, 'oranges': 1}}
>>> cart._contents
{'apples': 2, 'oranges': 1}
>>> Cart
<class '__main__.Cart'>
>>> cart.__dict__
{'_contents': {'apples': 2, 'oranges': 1}}
>>> order.__dict__
{'add': False, 'item': '', 'delete': False, 'quit': True}
>>> order.__dict__["quit"]
True
>>> ================================ RESTART ================================
>>>
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
q
<class '__main__.Cart'> {'_contents': {'oranges': 1, 'apples': 1}}
>>>
```

Ας δούμε τι περιέχει το καλάθι μας

- Υπάρχουν ειδικές methods και attributes που μας βοηθάνε.

```
class Cart:

    def __init__(self):
        self._contents = dict()

    def __repr__(self):
        return "{0} {1}".format(Cart, self.__dict__)
```

Κυρίως πρόγραμμα

```
cart = Cart()
order = Order()
order.get_input()

while not order.quit:
    cart.process(order)
    order = Order()
    order.get_input()

print(cart)
```

Python Modules

- Σχεδόν κάθε προγραμματιστικό περιβάλλον παρέχει τρόπο για ομαδοποίηση και οργάνωση κώδικα.
- Το έχουμε ήδη κάνει με τις συναρτήσεις και τις κλάσεις.
- Μπορούμε όμως να πετύχουμε ακόμα παραπάνω ομαδοποιώντας τον κώδικά μας σε διαφορετικά αρχεία, κάτι που ήδη είδαμε με τη JavaScript.
- Στην περίπτωση της Python αυτό γίνεται με τα modules και ήδη έχουμε χρησιμοποιήσει ένα module.

```
>>> ========== RESTART ==========
>>>
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
q
<class '__main__.Cart'> {'_contents': {'oranges': 1, 'apples': 1}}
>>> __name__
'__main__'
>>>
```

Sales.py

```
class Cart:

    def __init__(self):
        self._contents = dict()

    def __repr__(self):
        return "{0} {1}".format(Cart, self.__dict__)

    def process(self, order):
        if order.add:
            if not order.item in self._contents:
                self._contents[order.item] = 0
            self._contents[order.item] += 1
        elif order.delete:
            if order.item in self._contents:
                self._contents[order.item] -= 1
                if self._contents[order.item] <=0:
                    del self._contents[order.item]

class Order:

    def __init__(self):
        self.quit = False
        self.add = False
        self.delete = False
        self.item = None

    def get_input(self):
        print("[command] [item] (command is a to add, d to delete, q to quit)")
        line = input()

        command = line[:1]
        self.item = line[2:]

        if command == "a":
            self.add = True
        elif command == "d":
            self.delete == True
        elif command == "q":
            self.quit = True
```

Κυρίως πρόγραμμα

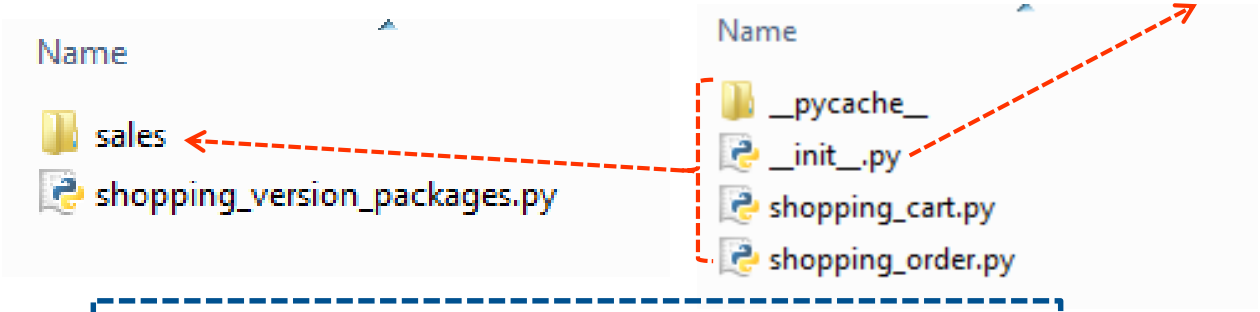
```
import sales

cart = sales.Cart()
order = sales.Order()
order.get_input()

while not order.quit:
    cart.process(order)
    order = sales.Order()
    order.get_input()

print(cart)
```

Προσέξτε το module το οποίο έχουμε.

```
>>> ================================ RESTART ================================
>>>
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
q
<class 'sales.Cart'> {'_contents': {'apples': 1, 'oranges': 1}}
>>>
```

Python Packages

- Όταν έχουμε κώδικα σε πολλαπλούς καταλόγους.
- Στην Python ένας υποκατάλογος γίνεται package.
- Όπως οι κλάσεις έχουν init μέθοδο, έτσι μπορεί να έχει και ένα module ή ένα package. Για το λόγο αυτό φτιάχνουμε ένα κενό αρχείο στον ίδιο φάκελο με το όνομα __init__.py



Κυρίως πρόγραμμα

```
import sales.shopping_cart, sales.shopping_order

cart = sales.shopping_cart.Cart()
order = sales.shopping_order.Order()
order.get_input()

while not order.quit:
    cart.process(order)
    order = sales.shopping_order.Order()
    order.get_input()

print(cart)
```