# Python – Κατανόηση της σημασιολογίας των αναφορών

# Κατανόηση σημασιολογίας αναφορών

- Η ανάθεση τιμής χειρίζεται αναφορές
  - Η x=y δεν δημιουργεί αντίγραφο του αντικειμένου που αναφέρεται η y.
  - Η x=y κάνει το x να αναφερθεί στο αντικείμενο που αναφέρεται η y.
- Πολύ χρήσιμο, αλλά απαιτείται προσοχή !!
- Παράδειγμα:

```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print b  [1, 2, 3, 4]
```

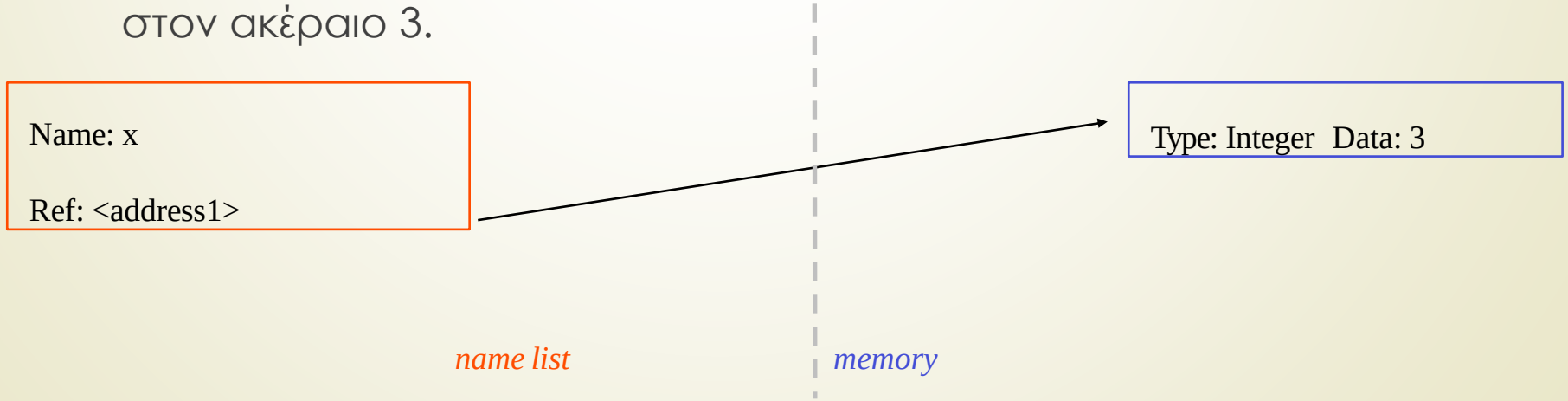
- Γιατί;

# Κατανόηση σημασιολογίας αναφορών II

- Είναι πολλά που συμβαίνουν όταν πληκτρολογούμε τον κώδικα:

  x=3

- Πρώτον, ένας ακέραιος 3 δημιουργείται και αποθηκεύεται στη μνήμη.
- Ένα όνομα x δημιουργείται.
- Μια αναφορά στη θέση μνήμης που είναι αποθηκευμένο το 3 ανατίθεται στο όνομα x.
- Έτσι όταν λέμε ότι η τιμή του x είναι 3 εννοούμε ότι το x αναφέρεται τώρα στον ακέραιο 3.



# Κατανόηση σημασιολογίας αναφορών III

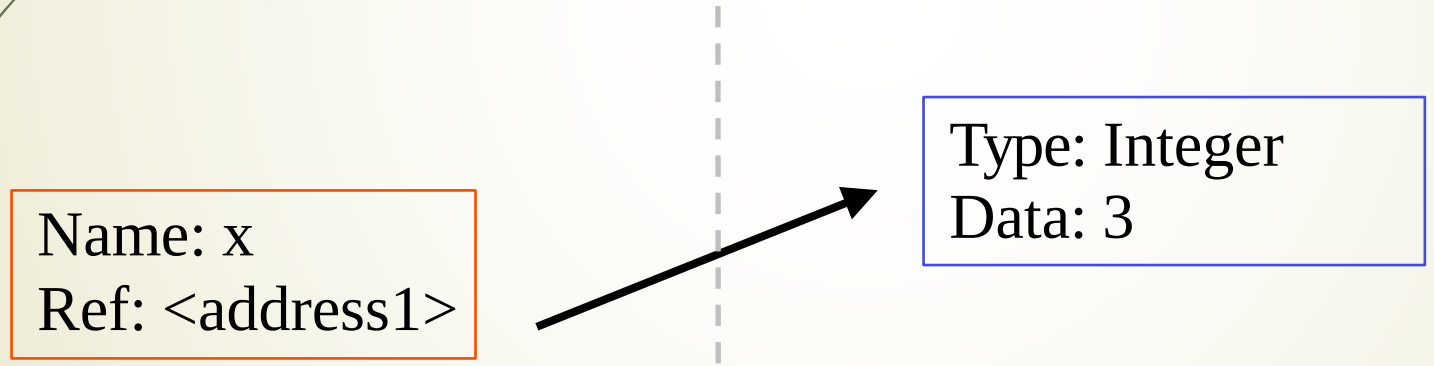
- Το δεδομένο 3 που δημιουργήθηκε είναι τύπου integer. Στην Python οι τύποι δεδομένων integer, float, και string (και tuple) είναι αμετάβλητοι.
- Αυτό δεν σημαίνει ότι δεν μπορούμε να αλλάξουμε την τιμή του x, αλλά αλλάζουμε αυτό που αναφέρεται το x.
- Παράδειγμα: μπορούμε να αυξήσουμε το x

```
>>> x=3
>>> x=x+1
>>> print x
>>> 4
```

# Κατανόηση σημασιολογίας αναφορών IV

- Αν αυξήσουμε το x, αυτό που πραγματικά συμβαίνει είναι:
  1. Αναζητείται η αναφορά του ονόματος x
  2. Ανακτάται η τιμή αυτής της αναφοράς

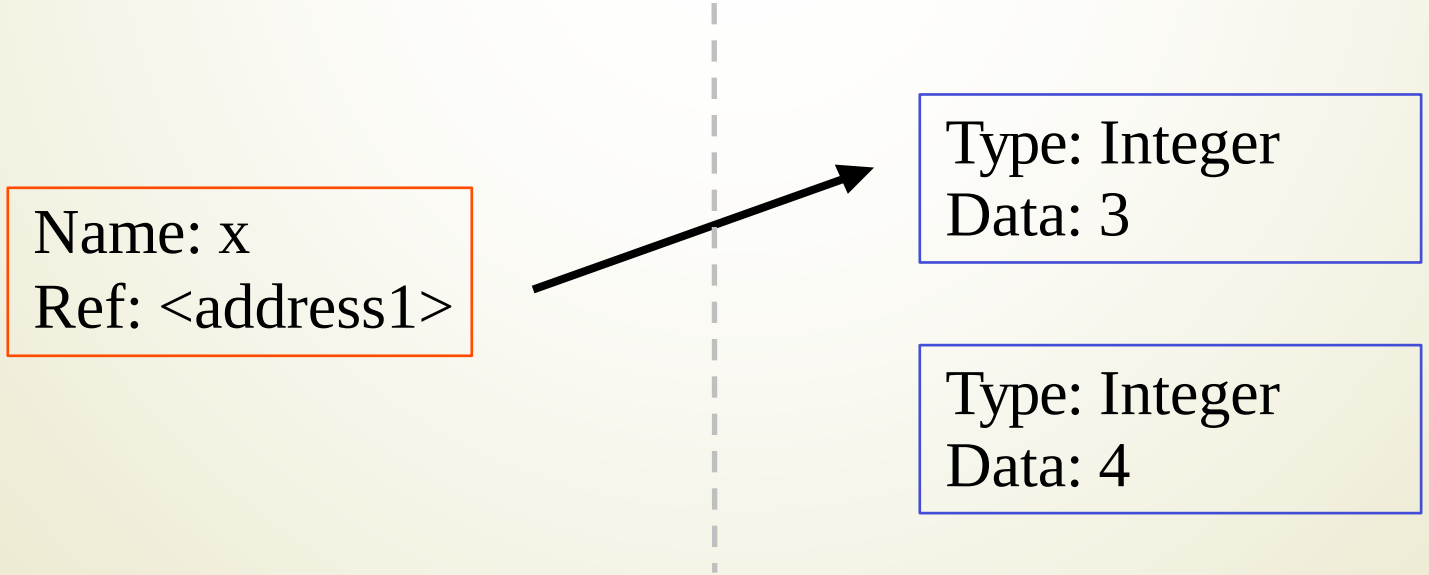
```
>>> x = x + 1
```



# Κατανόηση σημασιολογίας αναφορών IV

- Αν αυξήσουμε το x, αυτό που πραγματικά συμβαίνει είναι:
  1. Αναζητείται η αναφορά του ονόματος x
  2. Ανακτάται η τιμή αυτής της αναφοράς
  3. *Γίνεται ο υπολογισμός 3+1, παράγει το νέο στοιχείο δεδομένων **4** το οποίο ανατίθεται σε μία νέα θέση μνήμης με μία νέα αναφορά.*

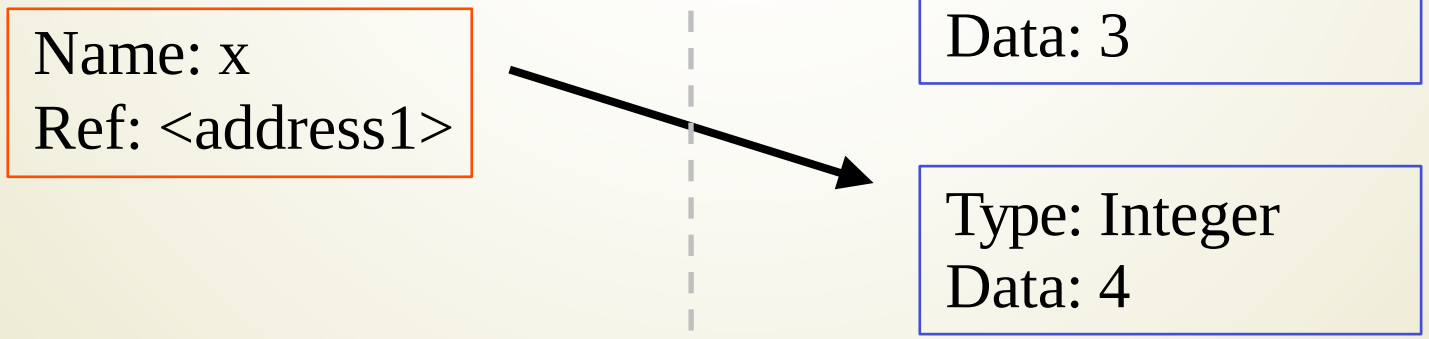
```
>>> x = x + 1
```



# Κατανόηση σημασιολογίας αναφορών IV

- Αν αυξήσουμε το x, αυτό που πραγματικά συμβαίνει είναι:
  1. Αναζητείται η αναφορά του ονόματος x
  2. Ανακτάται η τιμή αυτής της αναφοράς
  3. Γίνεται ο υπολογισμός 3+1, παράγει το νέο στοιχείο δεδομένων 4 το οποίο ανατίθεται σε μία νέα θέση μνήμης με μία νέα αναφορά.
  4. *Το όνομα **X** ιαλλάζει ώστε να δείχνει σε αυτή τη νέα αναφορά.*

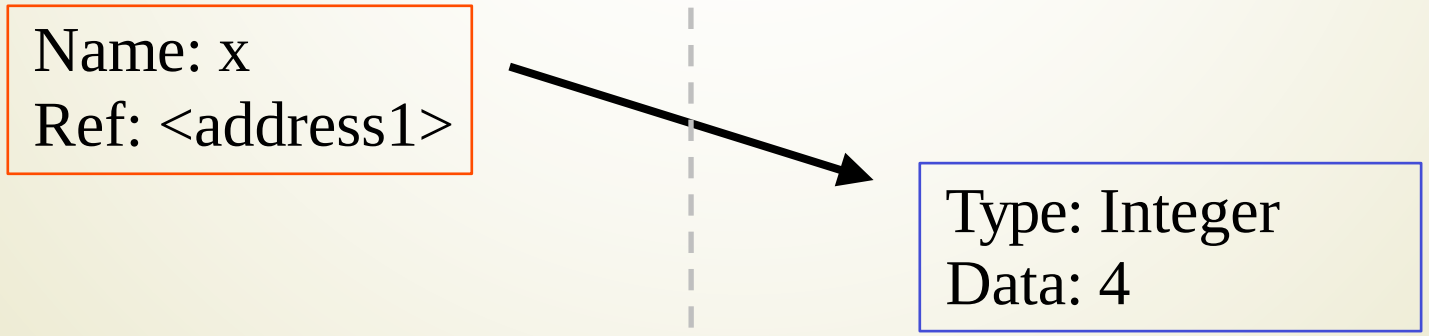
```
>>> x = x + 1
```



# Κατανόηση σημασιολογίας αναφορών IV

- Αν αυξήσουμε το x, αυτό που πραγματικά συμβαίνει είναι:
  1. Αναζητείται η αναφορά του ονόματος x
  2. Ανακτάται η τιμή αυτής της αναφοράς
  3. Γίνεται ο υπολογισμός 3+1, παράγει το νέο στοιχείο δεδομένων 4 το οποίο ανατίθεται σε μία νέα θέση μνήμης με μία νέα αναφορά.
  4. Το όνομα x ιαλλάζει ώστε να δείχνει σε αυτή τη νέα αναφορά.
  5. *Το παλαιό δεδομένο **3** απορρίπτεται ως «σκουπίδι» αν δεν υπάρχει όνομα που να αναφέρεται σε αυτό.*

```
>>> x = x + 1
```



# Ανάθεση τιμής 1

- **Άρα, για τους απλούς τύπους δεδομένων (integers, floats, strings), η ανάθεση τιμής συμπεριφέρεται όπως αναμενόταν:**

```
>>> x = 3     # Δημιουργεί το 3, το όνομα x αναφέρεται στο 3
>>> y = x     # Δημιουργεί το όνομα y, που αναφέρεται στο 3.
>>> y = 4     # Δημιουργεί αναφορά για το 4. Αλλάζει το y.
>>> print x   # Καμία επίπτωση στο x, ακόμη αναφέρεται στο 3.
3
```

# Ανάθεση τιμής 1

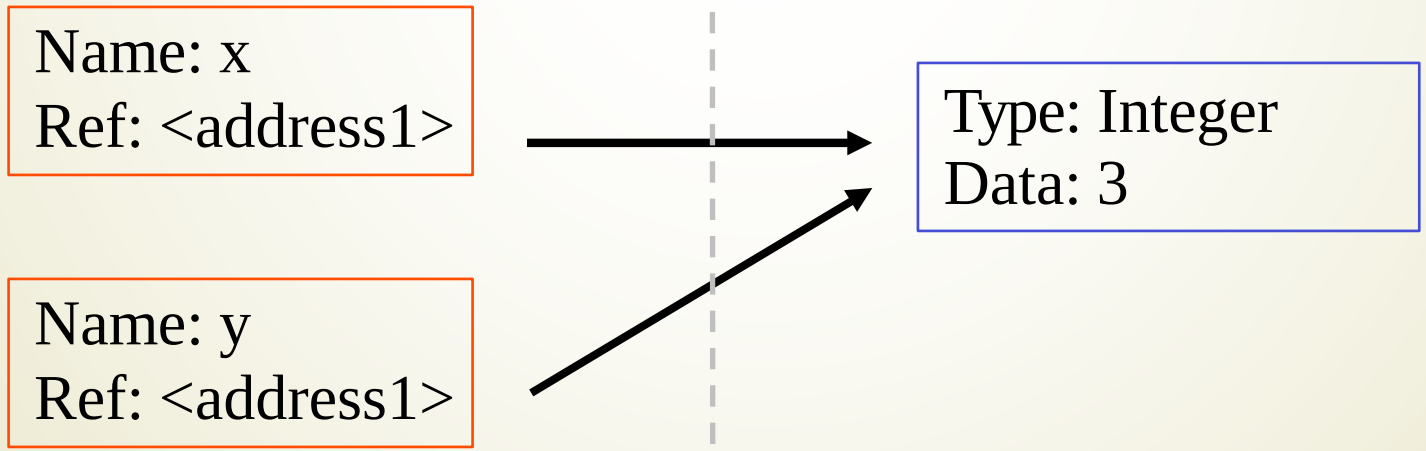
- **Άρα, για τους απλούς τύπους δεδομένων (integers, floats, strings), η ανάθεση τιμής συμπεριφέρεται όπως αναμενόταν:**

```
>>> x = 3      # Δημιουργεί το 3, το όνομα x αναφέρεται στο 3
>>> y = x      # Δημιουργεί το όνομα y, που αναφέρεται στο 3.
>>> y = 4      # Δημιουργεί αναφορά για το 4. Αλλάζει το y.
>>> print x    # Καμία επίπτωση στο x, ακόμη αναφέρεται στο 3.
3
```

# Ανάθεση τιμής 1

- **Άρα, για τους απλούς τύπους δεδομένων (integers, floats, strings), η ανάθεση τιμής συμπεριφέρεται όπως αναμενόταν:**

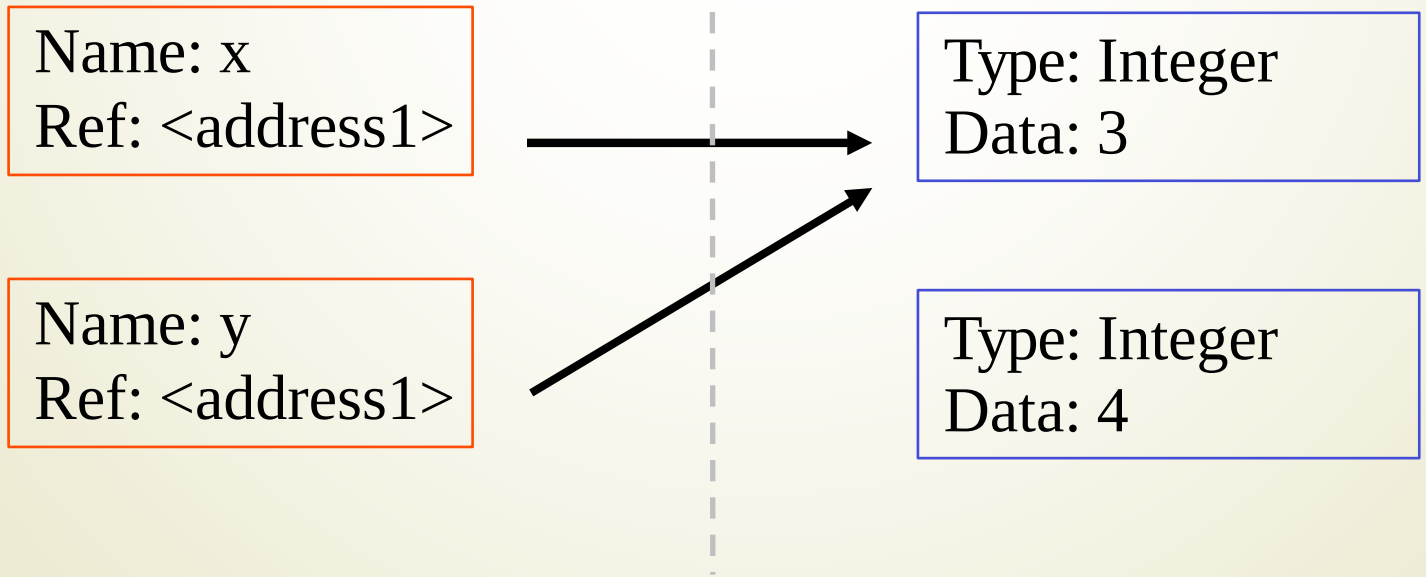
```
>>> x = 3      # Δημιουργεί το 3, το όνομα x αναφέρεται στο 3
>>> y = x      # Δημιουργεί το όνομα y, που αναφέρεται στο 3.
>>> y = 4      # Δημιουργεί αναφορά για το 4. Αλλάζει το y.
>>> print x    # Καμία επίπτωση στο x, ακόμη αναφέρεται στο 3.
3
```



# Ανάθεση τιμής 1

- **Άρα, για τους απλούς τύπους δεδομένων (integers, floats, strings), η ανάθεση τιμής συμπεριφέρεται όπως αναμενόταν:**

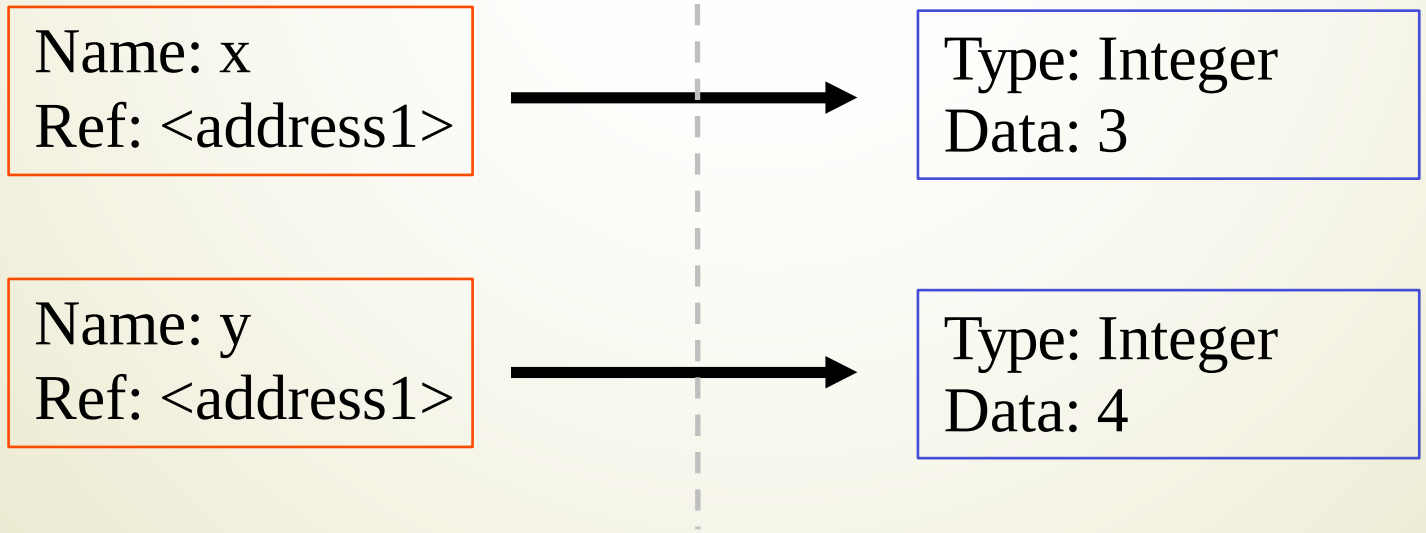
```
>>> x = 3      # Δημιουργεί το 3, το όνομα x αναφέρεται στο 3
>>> y = x      # Δημιουργεί το όνομα y, που αναφέρεται στο 3.
>>> y = 4      # Δημιουργεί αναφορά για το 4. Αλλάζει το y.
>>> print x    # Καμία επίπτωση στο x, ακόμη αναφέρεται στο 3.
3
```



# Ανάθεση τιμής 1

- **Άρα, για τους απλούς τύπους δεδομένων (integers, floats, strings), η ανάθεση τιμής συμπεριφέρεται όπως αναμενόταν:**

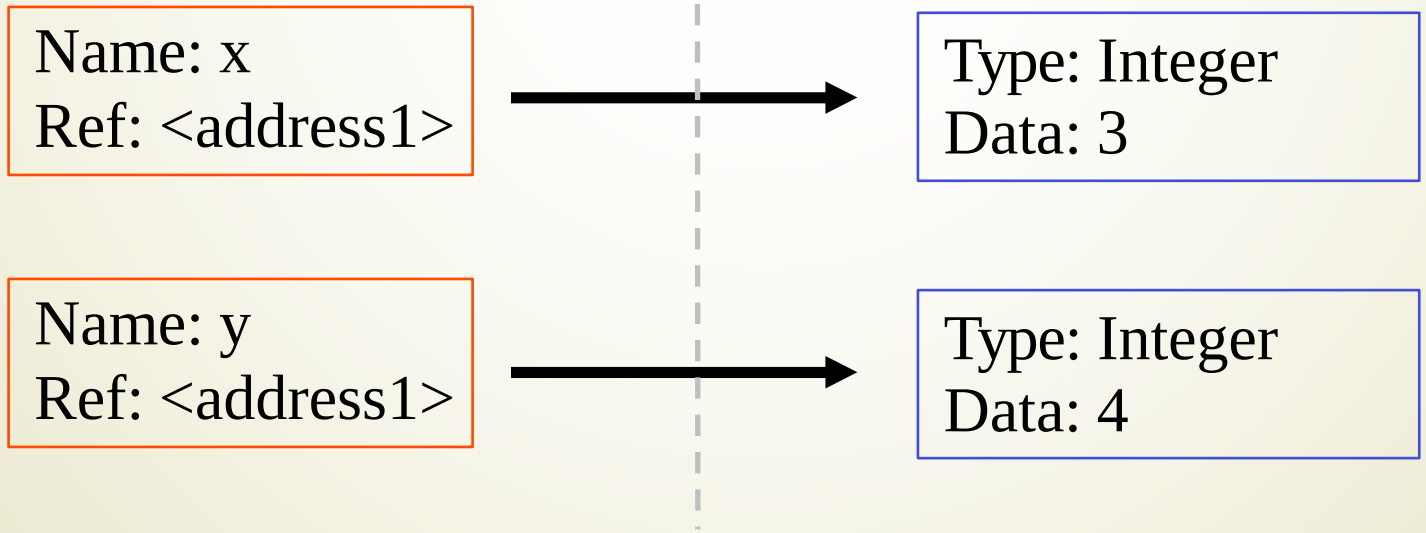
```
>>> x = 3      # Δημιουργεί το 3, το όνομα x αναφέρεται στο 3
>>> y = x      # Δημιουργεί το όνομα y, που αναφέρεται στο 3.
>>> y = 4      # Δημιουργεί αναφορά για το 4. Αλλάζει το y.
>>> print x    # Καμία επίπτωση στο x, ακόμη αναφέρεται στο 3.
3
```



# Ανάθεση τιμής 1

- **Άρα, για τους απλούς τύπους δεδομένων (integers, floats, strings), η ανάθεση τιμής συμπεριφέρεται όπως αναμενόταν:**

```
>>> x = 3      # Δημιουργεί το 3, το όνομα x αναφέρεται στο 3
>>> y = x      # Δημιουργεί το όνομα y, που αναφέρεται στο 3.
>>> y = 4      # Δημιουργεί αναφορά για το 4. Αλλάζει το y.
>>> print x    # Καμία επίπτωση στο x, ακόμη αναφέρεται στο 3.
3
```



# Ανάθεση τιμής 2

- **Για άλλους τύπους δεδομένων (λίστες, λεξικά, τύποι ορισμένοι από το χρήστη), η ανάθεση τιμής συμπεριφέρεται διαφορετικά:**
  - Οι τύποι αυτοί είναι «μεταβλητοί».
  - Όταν αλλάζουμε τα δεδομένα αυτά, αυτό γίνεται πάνω στη θέση.
  - Δεν γίνεται αντιγραφή αυτών σε νέα διεύθυνση μνήμης κάθε φορά.
  - Εάν γράψουμε y=x και έπειτα τροποποιήσουμε το y, τόσο το x όσο και το y αλλάζουν.

*Αμετάβλητα*

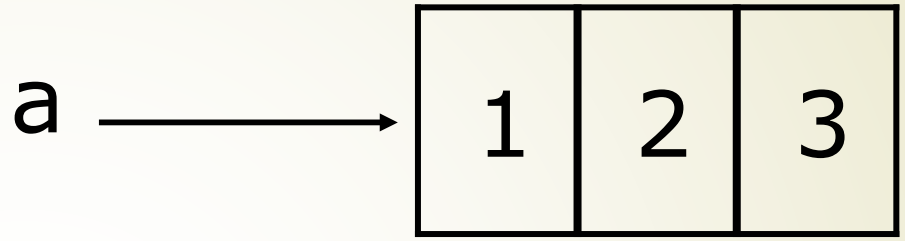
```
>>> x = 3
>>> y = x
>>> y = 4
>>> print x
3
```

*Μεταβλητά*

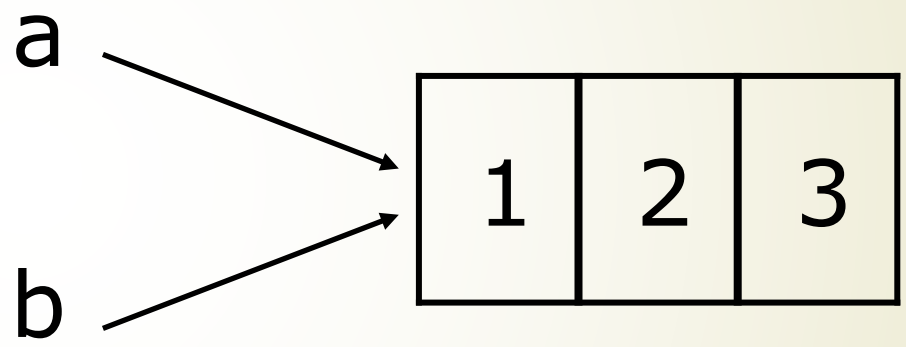
```
x = κάποιο μεταβλητό αντικείμενο
y = x
κάνε μια αλλαγή στο y
κοίταξε το x
Το x θα αλλάξει και αυτό.
```

# Γιατί; Αλλαγή λίστας

```
a = [1, 2, 3]
```




```
b = a
```




```
a.append(4)
```