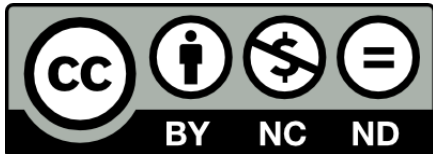


Εισαγωγή στην **Python**

3





Copyright

Το παρόν εκπαιδευτικό υλικό προσφέρεται ελεύθερα υπό τους όρους της άδειας Creative Commons:

- Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Όχι Παράγωγα Έργα 3.0.

Για να δείτε ένα αντίγραφο της άδειας αυτής επισκεφτείτε τον ιστότοπο

<https://creativecommons.org/licenses/by-nc-nd/3.0/gr/>

Στ. Δημητριάδης, 2015

Περιεχόμενα

- Αριθμητικοί τύποι δεδομένων
 - int, float, decimal, complex
- Αλφαριθμητικά
 - str
- I/O βασικές συναρτήσεις: **print** & **input**
- Μεταβλητές (variables)
 - Ονόματα μεταβλητών
 - Σύνδεση Ονομάτων με Αντικείμενα (object binding)
 - Ανάθεση τιμών (assignment)
- Δυναμική διαχείριση τύπων (Dynamic typing)
 - Οι συναρτήσεις type() & id()

Η Python έχει πολλούς τύπους δεδομένων

Object type	Example literals/creation
Numbers	1234, 3.1415, 3+4j, 0b111, Decimal(), Fraction()
Strings	'spam', "Bob's", b'a\x01c', u'sp\xc4m'
Lists	[1, [2, 'three'], 4.5], list(range(10))
Dictionaries	{'food': 'spam', 'taste': 'yum'}, dict(hours=10)
Tuples	(1, 'spam', 4, 'U'), tuple('spam'), namedtuple
Files	open('eggs.txt'), open(r'C:\ham.bin', 'wb')
Sets	set('abc'), {'a', 'b', 'c'}
Other core types	Booleans, types, None
Program unit types	Functions, modules, classes (Part IV , Part V , Part VI)
Implementation-related types	Compiled code, stack tracebacks (Part IV , Part VII)

Πηγή: Lutz, M. (2013). *Learning Python, 5th ed.*, O'Reilly: Cambridge



Αριθμητικοί Τύποι Δεδομένων

- integer (int)
- floating point (float)
- decimal
- complex

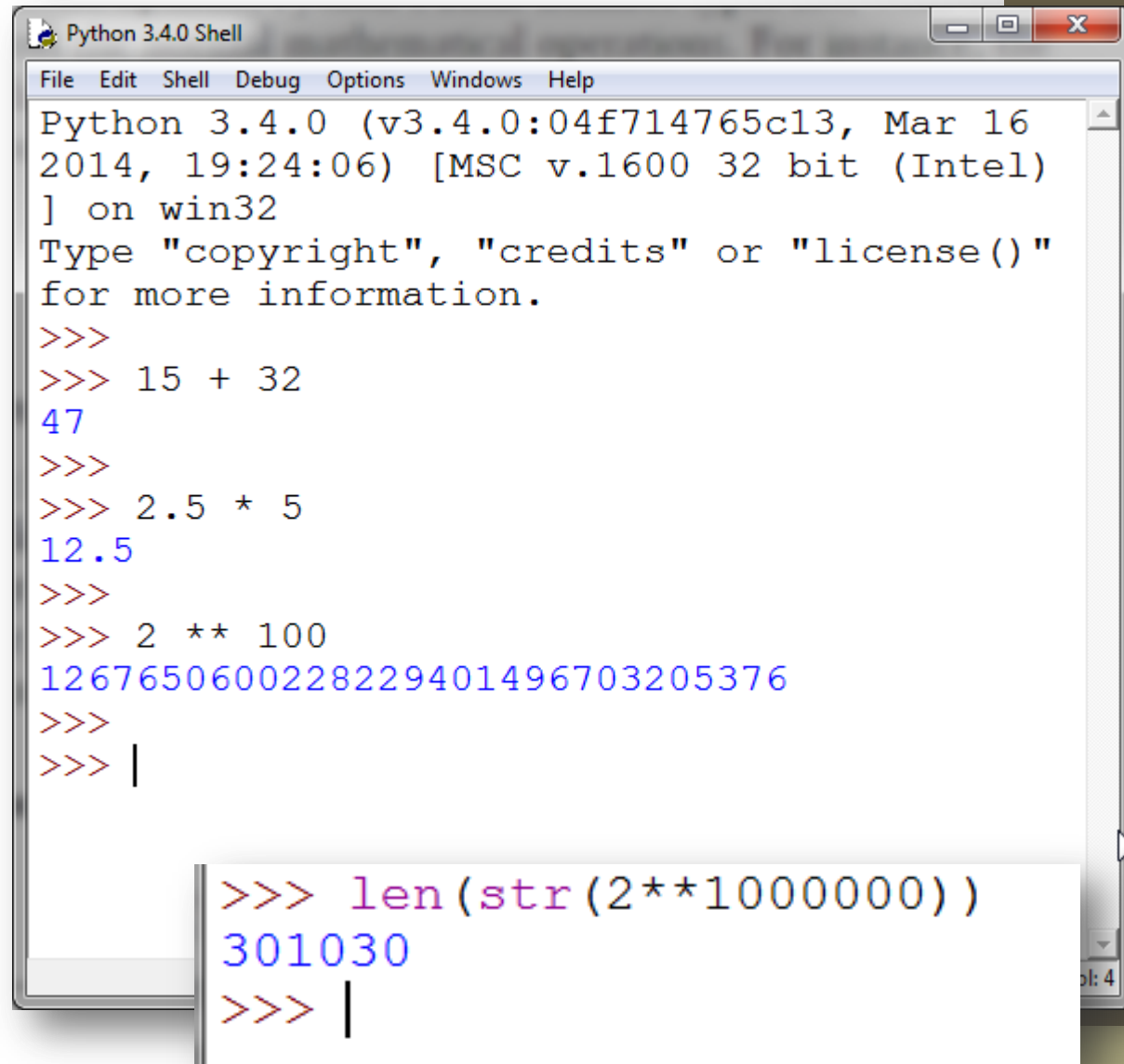
Numbers, Αριθμητικοί τύποι

- **integers** (int) ακέραιοι (χωρίς δεκαδικό τμήμα)
- **floating-point** (float) κινητής υποδιαστολής (με δεκαδικό τμήμα)
- **complex** μιγαδικοί (με 'φανταστικό' τμήμα)
- **decimal** καθορισμένης ακρίβειας
- **rational** με αριθμητή & παρονομαστή
- Περισσότεροι τύποι είναι διαθέσιμοι με χρήση πρόσθετων βιβλιοθηκών από τρίτους κατασκευαστές

int

integer, ακέραιος

- Στην Python 3.x οι ακέραιοι είναι θεωρητικά άπειρης ακρίβειας
- Η Python δεσμεύει δυναμικά τον απαραίτητο χώρο μνήμης για την αναπαράσταση οσοδήποτε μεγάλων ακεραίων



```
Python 3.4.0 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16
2014, 19:24:06) [MSC v.1600 32 bit (Intel)
] on win32
Type "copyright", "credits" or "license()"
for more information.
>>>
>>> 15 + 32
47
>>>
>>> 2.5 * 5
12.5
>>>
>>> 2 ** 100
1267650600228229401496703205376
>>>
>>> |
>>> len(str(2**1000000))
301030
>>> |
```

- Προσοχή στο χρόνο που χρειάζεται για υπολογισμούς με τέτοιους μεγάλους αριθμούς, πχ. ο 2 στην 1000000 δύναμη έχει 301030 ψηφία

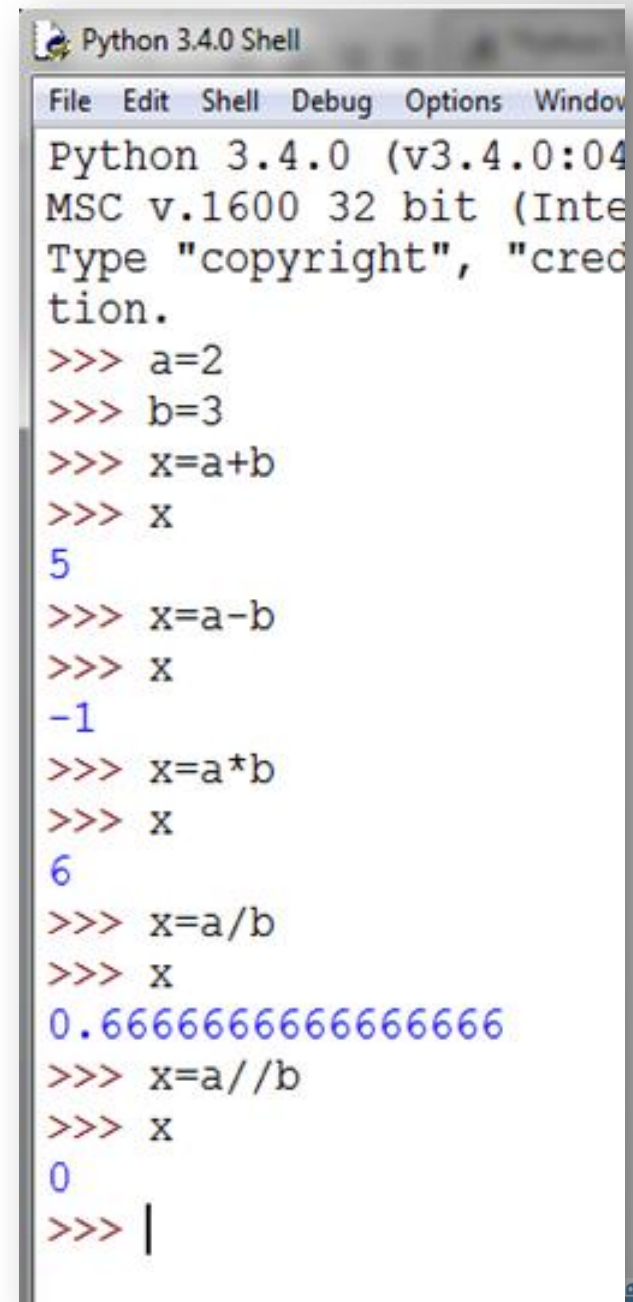
Integer Operators

Τελεστές ακεραίων

- Πρόσθεση **+**
- Αφαίρεση **-**
- Πολ/σμός *****
- Διαίρεση **/**
- Διαίρεση ακεραίων **//**

Είναι δυνατή η ομαδική ανάθεση τιμών

```
>>> a, b = 4, 8
>>> print(a, b)
4 8
>>> a, b, c = 4, 8, 16
>>> print(a, b, c)
4 8 16
>>>
```



```
Python 3.4.0 Shell
File Edit Shell Debug Options Window
Python 3.4.0 (v3.4.0:04
MSC v.1600 32 bit (Inte
Type "copyright", "cred
tion.
>>> a=2
>>> b=3
>>> x=a+b
>>> x
5
>>> x=a-b
>>> x
-1
>>> x=a*b
>>> x
6
>>> x=a/b
>>> x
0.6666666666666666
>>> x=a//b
>>> x
0
>>> |
```


Τελεστές διαίρεσης ακεραίων:

Διαφορά 2.x & 3.x

- Η **2.x** δεν έχει τελεστή διαίρεσης ακεραίων που να επιστρέφει δεκαδικό υπόλοιπο. Είτε με `'/'` είτε με `'//'` παίρνετε το ακέραιο πηλίκο.
- Για να έχετε επιστροφή δεκαδικού (float) πρέπει κάποιος από τους διαιρέτη ή διαιρετέο να είναι float

```
>>> 4/2
2.0
>>> 4//2
2
>>> 3/2
1.5
>>> 3//2
1
```

- Στην **3.x** οι τελεστές είναι σαφείς:
- `'/'`: Τελεστής για διαίρεση με υπόλοιπο (float)
- `'//'`: Τελεστής για διαίρεση ακεραίων (μόνον ακέραιο πηλίκο)

```
>>> 4/2
2
>>> 3/2
1
>>> 4//2
2
>>> 3//2
1
>>> 3/2.
1.5
>>> 4/2.
2.0
>>> 3./2
1.5
```

Από πού γεννιούνται οι int;

-1

Μια «κλεφτή» ματιά στις κλάσεις της Python

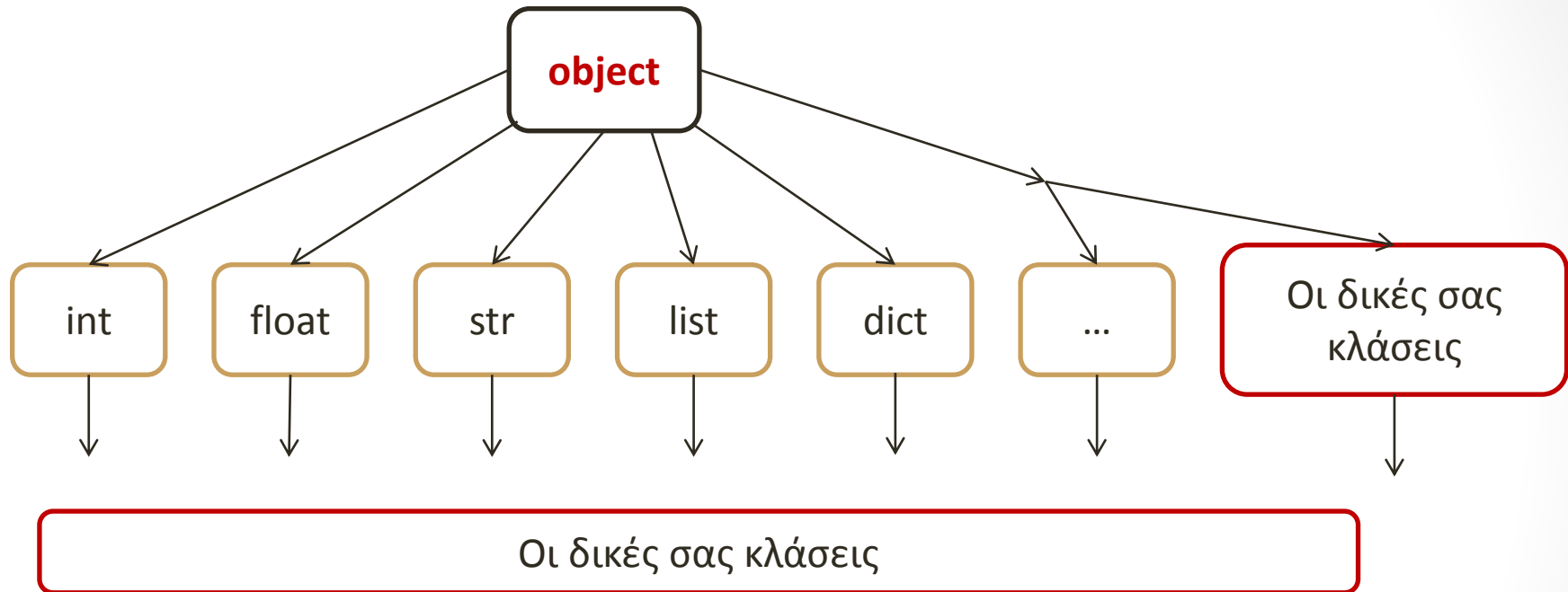
```
>>> type(1)
<class 'int'>
>>>
>>> isinstance(int, object)
True
>>>
>>> dir(int)
['__abs__', '__add__', '__and__', '__divmod__', '__doc__', '__eq__',
 '__float__', '__getattribute__', '__getitem__', '__getnewargs__',
 '__hash__', '__invert__', '__le__', '__lshift__', '__new__', '__or__',
 '__pos__', '__reduce_ex__', '__repr__', '__rpow__', '__rrshift__',
 '__rtruediv__', '__sizeof__', '__str__', '__subclasshook__',
 '__truediv__', '__bit_length__', 'conjugate']
>>>
>>> int.__add__(1, 2)
3
```

- Δοκιμάστε τον κώδικα στο κέλυφος
- Η τιμή 1 είναι ένα αντικείμενο τύπου **int**
- Η κλάση int είναι υποκλάση (κατώτερη) της ανώτερης **object**
- Οι ιδιότητες της int που κληρονομεί από την object
- Μπορείτε να κάνετε πρόσθεση καλώντας τη μέθοδο **__add__** της κλάσης int με ορίσματα τους δύο προσθετέους

Από πού γεννιούνται οι int;

-2

Μια «κλεφτή» ματιά στις κλάσεις της Python



- Ιεραρχία κλάσεων: όλες οι κλάσεις κληρονομούν από την ανώτερη **object**
- Οι διάφοροι τύποι δεδομένων υλοποιούνται ως κλάσεις (int, float, κλπ.)
- Κάθε τιμή (literal) είναι αντικείμενο-στιγμιότυπο της αντίστοιχης κλάσης
- Μπορείτε να γράψετε τις δικές σας κλάσεις στην ιεραρχία

κινητής υποδιαστολής, πραγματικοί

- Η Python ακολουθεί την προδιαγραφή **IEEE 754** για τους αριθμούς κινητής υποδιαστολής, δηλ. αριθμούς που περιλαμβάνουν:
- (α) Το **σημαντικό τμήμα** του αριθμού (**significand**)
- (β) Το **εκθετικό μέρος** (**exponent**) που καθορίζει το πόσα είναι τα δεκαδικά ψηφία

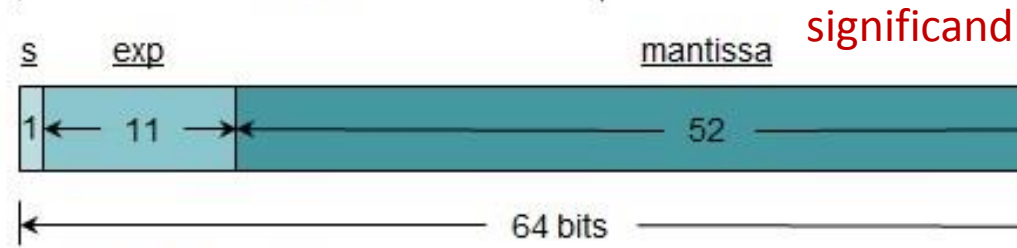
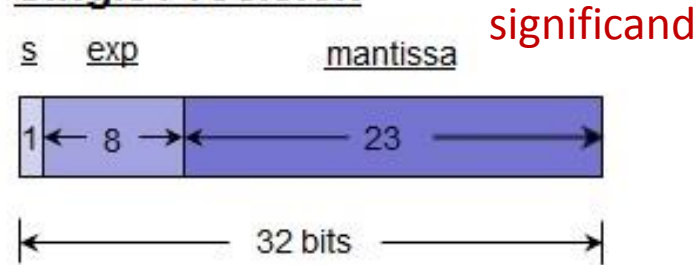
significand × **base** ^{exponent}

$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10^{-4}}_{\text{base}}^{\text{exponent}}$$

```
>>> 1e10
10000000000.0
>>> 5e-2
0.05
>>> 0.3e3
300.0
```

- Η προδιαγραφή IEEE 754 προβλέπει δύο κατηγορίες αριθμούς κινητής υποδιαστολής:
- (1) **Απλής** ακρίβειας (single precision)
- (2) **Διπλής** ακρίβειας (double precision)
- Στην τυπική CPython οι float αναπαρίστανται πάντοτε ως “double precision”

Single Precision



Double Precision

Format	Total bits	Significand bits	Exponent bits	Smallest number	Largest number
Single precision	32	23 + 1 sign	8	ca. $1.2 \cdot 10^{-38}$	ca. $3.4 \cdot 10^{38}$
Double precision	64	52 + 1 sign	11	ca. $5.0 \cdot 10^{-324}$	ca. $1.8 \cdot 10^{308}$

float η βιβλιοθήκη math

- Πράξεις μεταξύ ακεραίων και πραγματικών επιστρέφουν πάντοτε πραγματικό
- Ειδικότερες μαθηματικές ποσότητες (πχ. 'π', e) και συναρτήσεις είναι διαθέσιμες μέσω της ενσωματωμένης βιβλιοθήκης **math**
- Άλλες βιβλιοθήκες από τρίτους κατασκευαστές, πχ. η SciPy προσφέρουν διάφορες κλάσεις, μεθόδους και συναρτήσεις για επεξεργασία δεδομένων επιστημονικού ενδιαφέροντος (πχ. στατιστική)
- Δείτε [εδώ](#) περισσότερα για τη math

```
>>> 4 + 2.5
6.5
>>> import math
>>> math.pi
3.141592653589793
>>>
>>> math.e
2.718281828459045
>>>
>>> math.sqrt(97)
9.848857801796104
>>>
```

- Η ακρίβεια με την οποία αναπαρίστανται υπολογιστικά οι πραγματικοί είναι **περιορισμένη**
 - Αυτό ισχύει για όλες τις γλώσσες προγραμματισμού και έχει να κάνει με τους περιορισμούς του υλικού (hardware) αφού δεν υπάρχουν απεριόριστα ή πάρα πολλά bit μνήμης για αναπαράσταση δεκαδικών με πολλά ή άπειρα ψηφία
- Η περιορισμένη ακρίβεια σε περιπτώσεις μπορεί να δημιουργήσει προβλήματα στους υπολογισμούς
- *Παράδειγμα*
- $0.1 + 0.2$ δεν ισούται με 0.3
- Γιατί;
- Δείτε [εδώ](#) περισσότερα για τον περιορισμό της ακρίβειας στους float

```
>>> x = 0.1
>>> y = 0.2
>>>
>>> x+y
0.30000000000000004
>>>
>>> (x+y) == 0.3
False
>>>
```

- Οι πραγματικοί γενικά αναπαρίστανται προσεγγιστικά
- Παράδειγμα
- Ο δεκαδικός **0.125** υπολογίζεται ως $1/10 + 2/100 + 5/1000 = 0.125$
- Σε δυαδική μορφή αναπαρίσταται ως **0.001** = $0/2 + 0/4 + 1/8 = 0.125$
- Όμως στο σύστημα δυαδικής αναπαράστασης **δεν είναι δυνατό όλοι οι δεκαδικοί να αναπαρίστανται ακριβώς ως δυνάμεις του 2**
- Πχ. ο 0.1 στο δυαδικό έχει άπειρα ψηφία:
- 0.0001100110011001100110011001100110011001100110011...
- Οπότε η δεκαδική του τιμή είναι κι αυτή προσεγγιστική:
- 0.100000000000000000055511151231257827021181583404541015625...

- Πρακτικά η Python κάνει «έξυπνες» **στρογγυλοποιήσεις** αλλά αν αρχίσετε να κάνετε υπολογισμούς σας περιμένουν εκπλήξεις γιατί η αναπαράσταση **δεν είναι ακριβής**

```
>>> x=0.1
>>> y=0.2
>>> x
0.1
>>> 2*x
0.2
>>> x+y
0.30000000000000004
```


Ακρίβεια: οι απλές λύσεις

- ```
>>> round(x+y, 2)
0.3
>>> round(x+y)
0
>>> round(x+y, 5)
0.3
>>> round(x+y, 125)
0.30000000000000004
```

```
>>> E = 1e-3
>>> E
0.001
>>> (x+y) == 0.3
False
>>> abs((x+y)-0.3) < E
True
```

- Ο τύπος **Decimal** επιτρέπει τον καθορισμό της ακρίβειας αναπαράστασης ενός πραγματικού αριθμού
- Χρησιμοποιείται συνήθως σε **εμπορικές εφαρμογές** όπου οι πραγματικοί πρέπει να έχουν συγκεκριμένη ακρίβεια, πχ. **δύο δεκαδικών**

- Η συνήθης χρήση του είναι ως εξής:

- (α) Εισάγετε τη σχετική βιβλιοθήκη decimal
- (β) Καθορίζετε το επίπεδο ακρίβειας μέσω της getcontext()
- (γ) Στη συνέχεια οποιαδήποτε αριθμητική τιμή τη μετατρέπετε σε Decimal() και κάνετε πράξεις
- (δ) Το αποτέλεσμα έχει την προκαθορισμένη ακρίβεια που ορίσατε στο βήμα (β)

```
>>> import decimal as dc
>>> dc.getcontext().prec = 3
>>> dc.Decimal(0.1)+dc.Decimal(0.2)
Decimal('0.300')
```

# complex

# Μιγαδικοί

```
>>> x = 2.5 + 4j
>>> a = -2
>>> b = 0.5
>>> w = complex(a, b)
>>> x
(2.5+4j)
>>> w
(-2+0.5j)
```

- Οι μιγαδικοί είναι αριθμοί με πραγματικό (real) και φανταστικό (imaginary) τμήμα

```
>>> w.real
-2.0
>>> w.imag
0.5
>>> w.conjugate()
(-2-0.5j)
```

- Η cmath είναι βιβλιοθήκη με διάφορες συναρτήσεις σχετικές με τους μιγαδικούς

```
>>> import cmath
>>> cmath.phase(w)
2.896613990462929
>>> cmath.polar(x)
(4.716990566028302, 1.0121970114513341)
```

# συμβολοσειρές, αλφαριθμητικά, strings

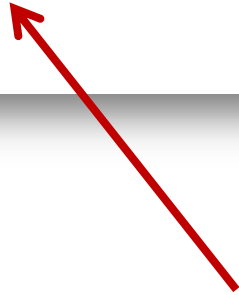
- Οι συμβολοσειρές (ή ‘αλφαριθμητικά’ εφόσον περιλαμβάνουν γράμματα και αριθμητικά ψηφία) χρησιμοποιούνται για την αποθήκευση:
- (α) **κειμένου** (πχ. ονόματα, κλπ.)
- (β) **συλλογή bytes** (πχ. τα bytes που περιέχονται σε ένα αρχείο εικόνας αντιστοιχούνται σε χαρακτήρες ASCII)
- Οι τιμές αλφαριθμητικών μπορούν να περικλείονται σε **μονά** ή **διπλά** εισαγωγικά
  - Πχ. ‘Spam’ ή “Spam” είναι αποδεκτά
  - Συνεπής χρήση: Να μην γίνεται σύγχυση μεταξύ των δύο τύπων εισαγωγικών
  - Όχι: ‘Spam”
- Ο τελεστής της **συνένωσης** (string concatenation) είναι το **+**

```
>>> s = 'spam'
>>> e = 'eggs'
>>> s+e
'spameggs'
>>> 'spam'+ 'eggs'
'spameggs'
>>>
```

```
>>> f = "foo"
>>> s = 'spam'
>>> f+s
'foospam'
>>>
```

- Τα αλφαριθμητικά είναι ένα χαρακτηριστικό παράδειγμα δομής **ακολουθίας** (sequence) στην Python
- Δηλ. αποτελεί μια **δεικτοδοτημένη (indexed)** συλλογή αντικειμένων
- Μια ακολουθία οργανώνει τα αντικείμενα που την αποτελούν σε σειρά από αριστερά προς τα δεξιά και η θέση τους προσδιορίζεται με έναν **δείκτη (index)**
- Ένα αλφαριθμητικό είναι ακολουθία χαρακτήρων

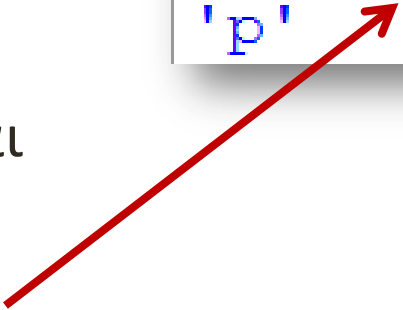
```
>>> s = 'spam'
>>> len(s)
4
```



- Η συνάρτηση **len()** επιστρέφει το μήκος (δηλ. πλήθος αντικειμένων μιας ακολουθίας)
- Για ένα αλφαριθμητικό επιστρέφει το πλήθος των χαρακτήρων που περιλαμβάνει

- Η Python είναι γλώσσα zero-indexed
- Δηλ. ένας δείκτης σε δομή ακολουθίας έχει πρώτη τιμή το '0'
- Η πρώτη θέση (πρώτος χαρακτήρας) σε ένα αλφαριθμητικό βρίσκεται στη θέση 0
- Προσέξτε τις **αγκύλες [ ]** που χρησιμοποιούνται για να προσδιοριστεί ο κάθε χαρακτήρας στο αλφαριθμητικό s

```
>>> s = 'spam'
>>> len(s)
4
>>> s[0]
's'
>>> s[1]
'p'
```



- O Guido van Rossum για το 0-based indexing

# str

- 4

- Προβλέψτε ποιες τιμές θα επιστρέψουν οι παραστάσεις δεξιά για το αλφαριθμητικό s;
- Πληκτρολογήστε τις στο κέλυφος και δείτε αν η πρόβλεψή σας ήταν σωστή
- Καταλαβαίνετε το μήνυμα λάθους που επιστρέφει η Python;

```
>>> s = 'spam'
>>> len(s)
4
>>> s[0]
's'
>>> s[1]
'p'
```

```
>>> s[len(s) - 1]
```

```
>>> s[len(s)]
```

```
Traceback (most recent call last):
 File "<pyshell#43>", line 1, in <module>
 s[len(s)]
IndexError: string index out of range
```



# Για τη συνέχεια...

- Γράψτε στον συντάκτη το παρακάτω απλό πρόγραμμα

```
print & input

my_name = input("Όνομα: ")
print('Γειά σου ' + my_name)
```

- Προσέξτε τη χρήση των συμβόλων:
- (α) **#** το σύμβολο για σχόλια
- (β) **+** το σύμβολο συνένωσης αλφαριθμητικών (Concatenation)
- (γ) τις εντολές **print** & **input**
- (δ) τον τρόπο γραφής της μεταβλητής **my\_name**
- (ε) τη χρήση μονών και διπλών εισαγωγικών
- *Γι αυτά θα μιλήσουμε στη συνέχεια*



# print() & input()



# print()

- 1

- Στην Python 3.x η **print** είναι **ενσωματωμένη συνάρτηση** (built-in function) με παραμέτρους που διαμορφώνουν την τελική μορφή εμφάνισης των αποτελεσμάτων

```
print([object, ...][, sep=' '][, end='\n'][, file=sys.stdout][, flush=False])
```

- Μέσα στις αγκύλες εμφανίζονται οι **κατ' επιλογήν παράμετροι** κατά την κλήση της print (μπορούν να παραλείπονται)
- Οι τιμές μετά το = είναι οι **εξ ορισμού τιμές** που δίνονται στις παραμέτρους κατά την κλήση
  - «εξ ορισμού» τιμές: αν δεν οριστούν αλλιώς τότε θεωρείται ότι περνούν αυτές οι τιμές στις παραμέτρους, κατά την κλήση της print



# print()

- 2

```
print([object, ...][, sep=' '][, end='\n'][, file=sys.stdout][, flush=False])
```

- **object** → λίστα παραμέτρων που περνάνε στην print για εμφάνιση
- **sep** → αλφαριθμητικό που λειτουργεί ως διαχωριστικό (separator) μεταξύ των παραμέτρων
- **end** → αλφαριθμητικό που καθορίζει τον τρόπο μετακίνησης του δείκτη μετά το τέλος της print,
  - το '\n' είναι η εξ ορισμού τιμή που δηλώνει 'νέα γραμμή' (new line)
- **file** → η μονάδα στην οποία κατευθύνεται η έξοδος της print
  - Το sys.stdout είναι η εξ ορισμού έξοδος (standard output) δηλ. η γραμμή εντολής ή 'κονσόλα' επικοινωνίας με τον υπολογιστή
- **flush** → λογική (boolean) παράμετρος που καθορίζει αν η έξοδος που παράγει η print θα εμφανιστεί άμεσα στη μονάδα εξόδου χωρίς ενδιάμεση αποθήκευση (buffering) (flush = 'True')
  - Στην πράξη δεν θα την χρειαστείτε παρά σε ειδικές περιπτώσεις



```
print([object, ...][, sep=' '][, end='\n'][, file=sys.stdout][, flush=False])
```

```
print() # Εμφανίζει μια κενή γραμμή
```

```
x = 'spam'
```

```
y = 99
```

```
z = ['eggs']
```

```
print(x, y, z) # Εμφανίζει τα x, y, z με τον εξορισμού διαχωριστή (κενό)
```

```
>>>
```

```
spam 99 ['eggs']
```

```
print(x, y, z, sep='') # Καταργεί τον διαχωριστή
```

```
>>>
```

```
spam99['eggs']
```

```
print(x, y, z, sep=', ') # Ορίζει νέο διαχωριστή
```

```
>>>
```

```
spam, 99, ['eggs']
```

```
print([object, ...][, sep=' '][, end='\n'][, file=sys.stdout][, flush=False])
```

*# Δεν αλλάζει γραμμή*

```
print(x, y, z, end='')
```

```
>>>
```

```
spam 99 ['eggs']
```

*# Οι print δίνουν έξοδο στην ίδια γραμμή*

```
print(x, y, z, end='')
```

```
print(x, y, z)
```

```
>>>
```

```
spam 99 ['eggs']spam 99 ['eggs']
```

*# Τοποθετεί στο τέλος '...' και αλλάζει γραμμή (\n)*

```
print(x, y, z, end='...\n')
```

```
>>>
```

```
spam 99 ['eggs']...
```

# Ο ρόλος του Backslash \ στην print() - 1

- Η 'όπισθοπλαγία' (**backslash**) είναι ο ειδικός χαρακτήρας (escape character) που διαμορφώνει:
- (α) τον **τρόπο γραφής** της print
- (β) την **έξοδο** που δίνει η print

```
a = 1
b = 2
c = 3
sm = a+b+c
dif=a-b-c
gin=a*b*c
pil=a/b/c

print(a, b, c, sm, dif, gin, pil)

print(a, b, c, \
 sm, dif, gin, pil)
```

- Χρησιμοποιήστε backslash για να **αναλύσετε** μια μεγάλη σε μήκος γραμμή/εντολή σε **περισσότερες γραμμές**
  - \* η δυνατότητα αυτή ισχύει για κάθε γραμμή ανεξάρτητα από το είδος κώδικα που περιλαμβάνουν

- Οι δύο print στο παράδειγμα είναι **ισοδύναμες**
  - Προσοχή: μην εισάγετε κενά ή tabs μετά το backslash

# Ο ρόλος του Backslash \ στην print() - 2

- Χρησιμοποιήστε backslash για να αλλάξετε **τη λειτουργία ορισμένων ειδικών χαρακτήρων** στην print (όπως δείχνει ο πίνακας)



| Escape Character | What Is Actually Printed |
|------------------|--------------------------|
| \\               | Backslash (\)            |
| \'               | Single quote (')         |
| \"               | Double quote (")         |
| \n               | Newline                  |
| \t               | Tab                      |

- Συγκρίνετε την print δεξιά με την έξοδο που παράγει (δεξιά κάτω)

```
x=1
y=2
z=3
print(x, y, '\'', '\t', z)
```

```
>>>
1 2 ' 3
>>>
```

# `print()` **docstring**: αλφαριθμητικά σε πολλές γραμμές

- Στην ειδική περίπτωση που θέλουμε να εισάγουμε ή να εμφανίσουμε ένα **αλφαριθμητικό** οργανωμένο **σε πολλές γραμμές** μπορούμε να το δηλώσουμε μέσα σε **τριπλά εισαγωγικά** (όπως στον κώδικα δεξιά για τη μεταβλητή `contents`)

```
contents='''
Περιεχόμενα
1) Κεφάλαιο 1
2) Κεφάλαιο 2
3) Κεφάλαιο 3
'''

a=1
b=2
print('Δεδομένα: \t',a,'\t',b,'\n')
print('Δεδομένα: \t a \t b \n')
print(contents)
```

- Γράψτε στον συντάκτη και τρέξτε τον κώδικα του παραδείγματος
- Κάντε μια εκτίμηση για το τι θα παρουσιάζει στην οθόνη και μετά τρέξτε το πρόγραμμά σας



- Κώδικας:

```
contents='''
Περιεχόμενα
1) Κεφάλαιο 1
2) Κεφάλαιο 2
3) Κεφάλαιο 3
'''

a=1
b=2
print('Δεδομένα: \t',a, '\t',b, '\n')
print('Δεδομένα: \t a \t b \n')
print(contents)
```

- Έξοδος:

```
Δεδομένα: 1 2

Δεδομένα: a b

Περιεχόμενα
1) Κεφάλαιο 1
2) Κεφάλαιο 2
3) Κεφάλαιο 3
```

# print() Μορφοποίηση εξόδου

-1

- Μορφοποίηση με την τεχνική “string modulo”
- (παλιότερη τεχνική)

```
print("Κωδικός: %5d, Τιμή μονάδας: %8.2f" % (745, 45.752))
```

Κωδικός: 745, Τιμή μονάδας: 45.75

Ομάδα τιμών

- Οι placeholders %5d και %8.2f καθορίζουν τις θέσεις όπου θα εμφανιστούν οι τιμές της ομάδας που ακολουθεί
- Δείτε άλλους [string formatting characters](#)

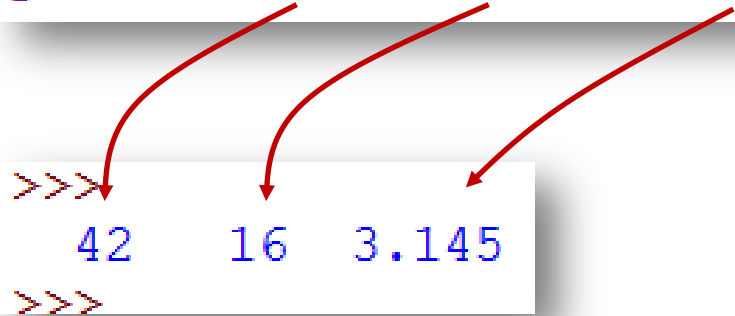
# print() Μορφοποίηση εξόδου

-2

- Μορφοποίηση με κλήση της μεθόδου **format()**
- (νεότερη τεχνική)

```
print('{:4d} {:4d} {:6.3f}'.format(42, 16, 3.145))
```

```
>>> 42 16 3.145
>>>
```



```
print('{2:8.4f} {1:6d} {0:3d}'.format(42, 16, 3.145))
```

```
>>> 3.1450 16 42
>>>
```



```
αλφα = 16
πι = 3.14
```

```
print('{2:8.4f} {1:6d} {0:3d}'.format(42, αλφα, 2*πι))
```

```
>>>
6.2800 16 42
>>>
```

- Η ομάδα τιμών μπορεί να περιλαμβάνει μεταβλητές ή και εκφράσεις
- Στην 3.x η κωδικοποίηση είναι **Unicode** και μπορείτε να χρησιμοποιήσετε όποιους χαρακτήρες θέλετε για τα ονόματα των δεδομένων

# print() Δυαδική, Οκταδική, & Δεκαεξαδική αναπαράσταση -4

```
a = 15
b = 1245
x = 3.14519
y = 23.45678
```

```
print('{2:6d} {2:6o} {2:6x} {2:6b}'.format(x, y, a, b))
```

```
>>>
15 17 f 1111
>>>
```

- **b** → Δυαδική
- **o** → Οκταδική
- **x** (ή **X**) → Δεκαεξαδική

```
>>> a = 15
>>>
>>> 0b1111
15
>>> 0xf
15
>>> 0o17
15
```

```
>>> a = 15
>>>
>>> bin(a)
'0b1111'
>>> hex(a)
'0xf'
>>> oct(a)
'0o17'
```

# print() ...σαν άσκηση

- Δημιουργήστε 4 μεταβλητές ως εξής:
  - Ακέραιες: a, b
  - Πραγματικές: x, y
- Στη συνέχεια γράψτε τις αντίστοιχες print() εφαρμόζοντας την τεχνική με .format() που δείξαμε, για να παρουσιάσετε στην οθόνη τις τιμές των μεταβλητών δίνοντας θέσεις:
  - Ακέραιες: 6
  - Πραγματικές: 8 με δεκαδικό μέρος 3
- Αλλάξτε τις τιμές των θέσεων και δείτε πώς επηρεάζεται η εμφάνιση των τιμών
- Εμφανίστε τις τιμές με δυαδική, οκταδική, δεκεξαδική μορφή

# input( )

- 1

- Η **συνάρτηση input( )** αποτελεί τον απλούστερο τρόπο για ανάγνωση της εισόδου που πληκτρολογεί ο χρήστης στην γραμμή εντολών
- Η input **περιμένει** μέχρι να πατηθεί Enter από τον χρήστη και διαβάζει τα δεδομένα εισόδου ως **αλφαριθμητικό (string)**
- Συνήθως η input περιλαμβάνει το μήνυμα προς τον χρήστη για την εισαγωγή δεδομένων

```
print('Όνομα: ')\nname = input()\n\nage = input('Ηλικία: ')\nprint(name, age)
```



# input()

- 2

- Στην Python 3 η input() επιστρέφει πάντοτε **αλφαριθμητικό (str)**
- Ένα συνηθισμένο λάθος για τον αρχάριο προγραμματιστή είναι να εισάγει αριθμητικά δεδομένα με την input χωρίς να μετατρέψει τον τύπο τους

```
Το x ορίζεται ως αλφαριθμητικό
x = input('Τιμή X: ')
```

```
Το x ορίζεται ως ακέραιος
x = int(input('Τιμή X: '))
```



# int(), float(), str()

- Χρησιμοποιήστε τις συναρτήσεις
- **int( ), float( ), str( )**
- Σε συνδυασμό με την input() για να **μετατρέψετε τον τύπο** της μεταβλητής σε ακέραιο, κινητής υποδιαστολής ή αλφαριθμητικό αντίστοιχα

```
Το x ορίζεται ως αλφαριθμητικό
x = input('Τιμή X: ')

Η int επιστρέφει ακέραιο
x = int(input('Τιμή X: '))

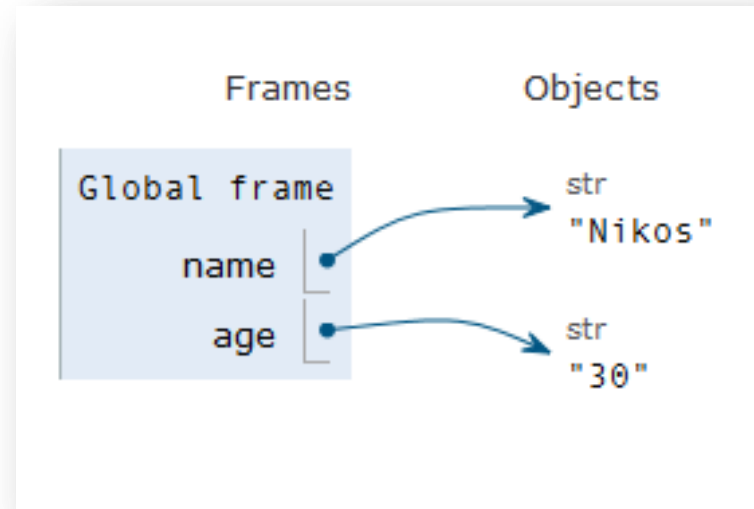
Η float επιστρέφει αριθμό κιν. υποδιαστολής
x = float(input('Τιμή X: '))

Το a μετατρέπεται σε αλφαριθμητικό
a = 15.4
a = str(a)
```

# Πώς ακριβώς λειτουργεί η input();

- Όταν εκτελείται η input() δημιουργεί **αντικείμενα τύπου str** στο χώρο μνήμης του προγράμματος και τα **συνδέει με τους αντίστοιχους αναγνωριστές** (ονόματα)

```
name = input('Όνομα: ')\nage = input('Ηλικία: ')
```



# Μεταβλητές (variables)

- Ονόματα μεταβλητών
- Σύνδεση Ονομάτων με Αντικείμενα (Object binding)
- Ανάθεση τιμών (assignment)
- Δυναμική διαχείριση τύπων (Dynamic typing)
- Οι συναρτήσεις `type( )` & `id( )`

# Ονόματα μεταβλητών

- 1

- Στην Python μια μεταβλητή είναι πρώτα απ' όλα ένα **όνομα** (name), δηλ. είναι ένας **αναγνωριστής** (identifier) για ένα αντικείμενο μέσα στον κώδικα.
- Απλοί κανόνες ονοματοδοσίας:

- (1) **Σύνταξη**:

*(underscore or letter) + (any number of letters, digits, or underscores)*

- Πχ. `_spam`, `spam`, `Spam_1` → **αποδεκτά** ονόματα
  - `1_Spam`, `spam$`, `@#!` → **μη** αποδεκτά ονόματα
- (2) Τα ονόματα είναι **'Case sensitive'**
    - Πχ. το `SPAM` **δεν** είναι το ίδιο με το `spam` ή το `Spam`

# Ονόματα μεταβλητών

- 2

- (3) **Δεσμευμένες** λέξεις **δεν** χρησιμοποιούνται ως ονόματα

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

Δεσμευμένες λέξεις στην Python 3.x



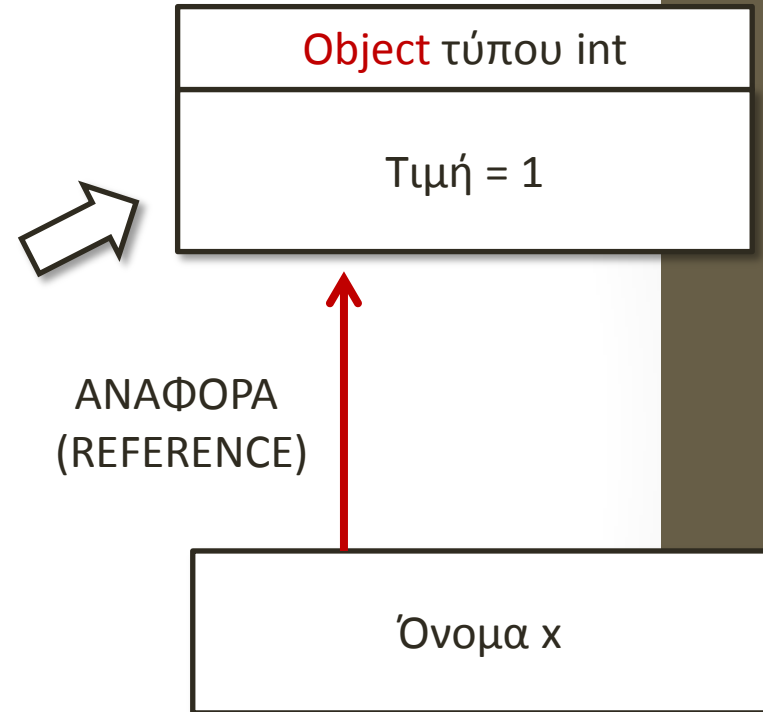
- (4) Υπάρχουν **δεσμευμένες** λέξεις με διπλό underscore στην αρχή και το τέλος που έχουν ιδιαίτερη σημασία για τον διερμηνέα

Πχ. `__init__`, `__name__` κ.ά

- Την ιδιαίτερη σημασία αυτών των αναγνωριστών θα τη συζητήσουμε στην ενότητα του αντικειμενοστρεφούς προγραμματισμού

# Αναφορές Ονομάτων σε Αντικείμενα - 1

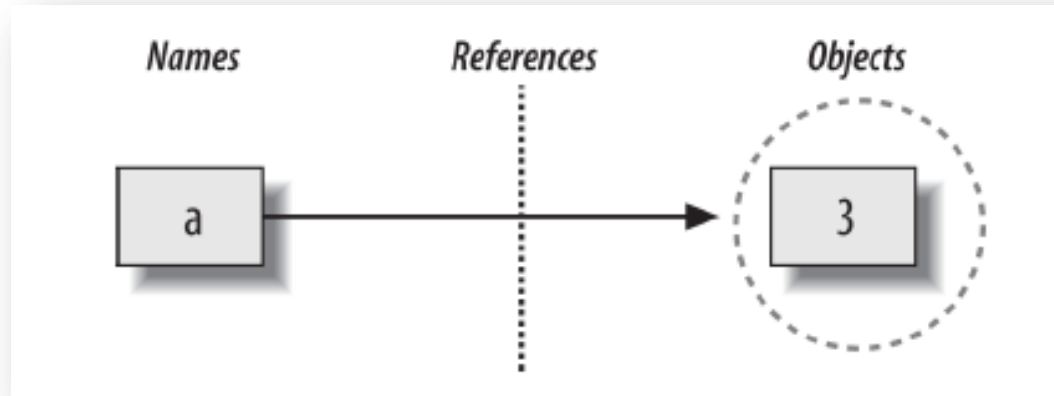
- Τι συμβαίνει όταν γράφουμε: **x = 1**
- (α) **Object**: Δημιουργείται στη μνήμη ένα αντικείμενο τύπου **int** με τιμή 1
- (β) **Binding** (σύνδεση): Το όνομα 'x' συνδέεται με το αντικείμενο που έχει τιμή '1'



# Αναφορές Ονομάτων σε Αντικείμενα

- 2

**a = 3**



- Στην εικόνα φαίνεται σχηματικά η **σύνδεση** (binding) του ονόματος **a** με το αντικείμενο με τιμή 3, όταν εκτελείται η παραπάνω εντολή ανάθεσης
- Η σύνδεση αυτή ονόματος – αντικειμένου καλείται απλά "**αναφορά**" (reference) και υλοποιείται με τη μορφή ενός δείκτη στη μνήμη
- Δηλ. τα ονόματα μεταβλητών αποτελούν απλές **αναφορές** προς αντικείμενα (objects)

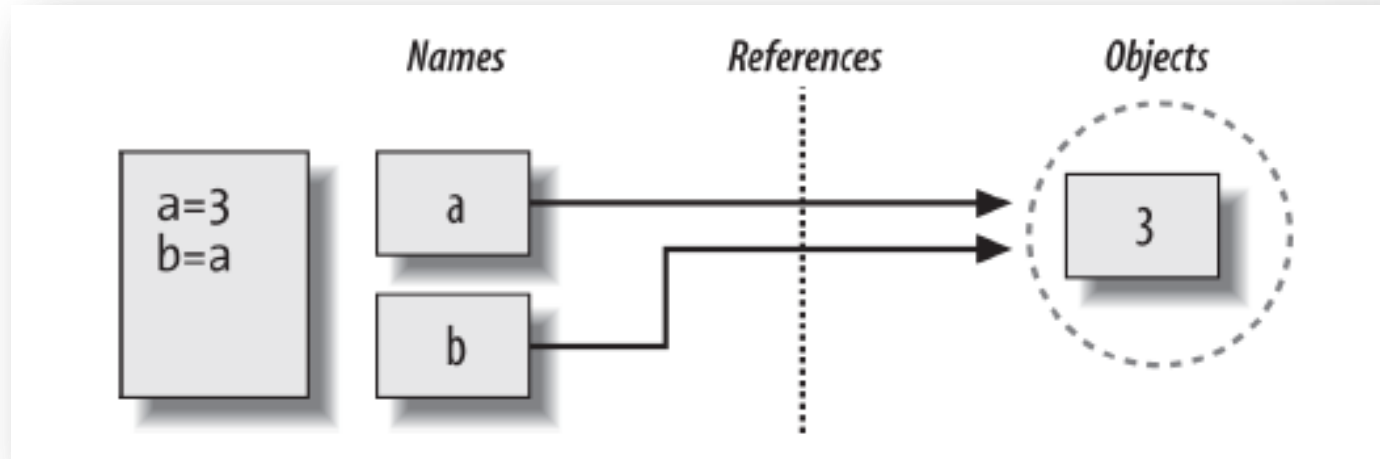


# Αναφορές Ονομάτων σε Αντικείμενα

- 3

**a = 3**

**b = a**



Πηγή: Lutz, M. (2013). *Learning Python*, 5<sup>th</sup> ed., O'Reilly: Cambridge

- Στην περίπτωση αυτή και τα δύο ονόματα 'δείχνουν' προς το ίδιο αντικείμενο
- Η περίπτωση αυτή όπου πολλαπλά ονόματα αναφέρονται στο ίδιο αντικείμενο-τιμή ονομάζεται "**διαμοιρασμένη αναφορά** ή αντικείμενο" (*shared reference* ή *shared object*)

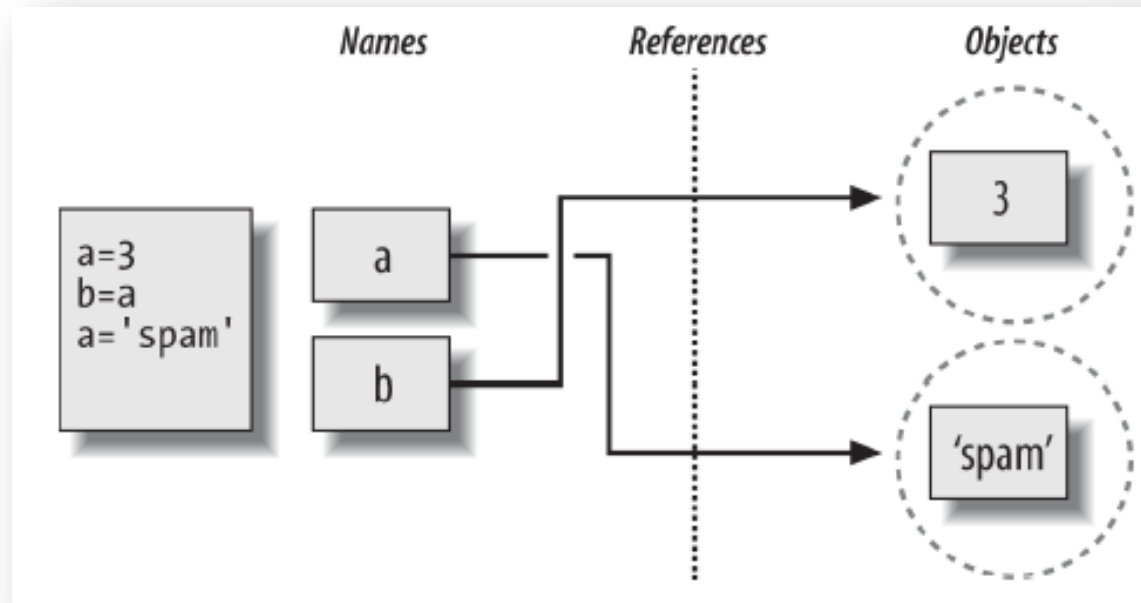
# Αναφορές Ονομάτων σε Αντικείμενα

- 4

**a = 3**

**b = a**

**a = 'spam'**



- Στην περίπτωση αυτή η αναφορά του ονόματος **a** **ανακατευθύνεται** προς το αλφαριθμητικό αντικείμενο με τιμή `'spam'`
- Παρατηρήστε ότι η αναφορά του ονόματος **b** στο αντικείμενο `3` δεν επηρεάζεται

# Αναφορές Ονομάτων σε Αντικείμενα

## Ερωτήσεις

- 5

- Εξηγήστε τι συμβαίνει στις παρακάτω περιπτώσεις κώδικα, όσον αφορά τις αναφορές των ονομάτων σε αντικείμενα
  - Σχεδιάστε σχήματα παρόμοια όπως στις προηγούμενες διαφάνειες

- (1)

**a = 3**

**b = a**

**a = a + 2**

- (2)

**a = 3**

**b = a**

**a = b + 2**

- (3)

**a = 3**

**b = 'spam'**

**a = b + str(2)**

- (4)

**a = 3**

**b = 4**

**x = a + b**

# Συμπερασματικά για την Αναφορά Ονομάτων σε Αντικείμενα

- Στην Python σχετικά με τις μεταβλητές ισχύει:
- **Ονόματα Μεταβλητών:** Καταλαμβάνουν τη δική τους περιοχή στη μνήμη (namespace = χώρος ονομάτων) και αποτελούν αναφορές για τα αντικείμενα με τα οποία είναι συνδεδεμένα
- **Αντικείμενα:** δεσμεύουν τον απαραίτητο χώρο στη μνήμη για αποθήκευση των δεδομένων που τα συνθέτουν
- **Αναφορές (References):** αυτόματα δημιουργούμενοι δείκτες από τα ονόματα μεταβλητών προς τα αντίστοιχα αντικείμενα

- Γενικά στην Python κατά την ανάθεση τιμών σε μεταβλητές ισχύουν τα παρακάτω:
- (1) Μια **εντολή ανάθεσης δημιουργεί αναφορά (δείκτη) σε αντικείμενα** στη μνήμη
  - Όπως εξηγήσαμε στα προηγούμενα
- (2) Τα **ονόματα μεταβλητών δημιουργούνται την πρώτη φορά που θα γίνει ανάθεση** τιμής στο όνομα αυτό
  - Πχ. η ανάθεση  $x = 1$  δημιουργεί και το όνομα/μεταβλητή  $x$

- (3) Πριν χρησιμοποιήσετε ένα όνομα μεταβλητής σε μία έκφραση **πρέπει να του έχετε αναθέσει τιμή**
  - Πχ.
  - `myvar = 1`
  - `print(Myvar)`
  - Ο παραπάνω κώδικας θα δώσει λάθος καθώς η μεταβλητή `Myvar` είναι διαφορετική από την `myvar` και δεν της έχει ανατεθεί τιμή
- (4) Ορισμένες λειτουργίες **εκτελούν αναθέσεις τιμής σιωπηρά (implicitly)**
  - Πχ. στη μεταβλητή ενός βρόχου `for` γίνεται ανάθεση τιμής κατά την εκτέλεση του βρόχου

| Operation                                 | Interpretation                                                        |
|-------------------------------------------|-----------------------------------------------------------------------|
| <code>spam = 'Spam'</code>                | Basic form                                                            |
| <code>spam, ham = 'yum', 'YUM'</code>     | Tuple assignment (positional)                                         |
| <code>[spam, ham] = ['yum', 'YUM']</code> | List assignment (positional)                                          |
| <code>a, b, c, d = 'spam'</code>          | Sequence assignment, generalized                                      |
| <code>a, *b = 'spam'</code>               | Extended sequence unpacking (Python 3.X)                              |
| <code>spam = ham = 'lunch'</code>         | Multiple-target assignment                                            |
| <code>spams += 42</code>                  | Augmented assignment (equivalent to <code>spams = spams + 42</code> ) |

Πηγή: Lutz, M. (2013). *Learning Python*, 5<sup>th</sup> ed., O'Reilly: Cambridge

- Αποδεκτές μορφές εντολών ανάθεσης
- **ΕΡΩΤΗΣΗ:** Εξηγήστε ποια τιμή θα έχουν οι μεταβλητές στο αριστερό μέρος μετά την εκτέλεση της κάθε εντολής ανάθεσης τιμής

# Απαντήσεις

- 1

```
spam = 'Spam'
```

spam ← 'Spam'

```
spam, ham = 'yum', 'YUM'
```

spam ← 'yum'

ham ← 'YUM'

```
[spam, ham] = ['yum', 'YUM']
```

spam ← 'yum'

ham ← 'YUM'





# Απαντήσεις

- 2

```
a, b, c, d = 'spam'
```

a ← 's'

b ← 'p'

c ← 'a'

d ← 'm'

```
a, *b = 'spam'
```

a ← 's'

b ← 'pam'

```
spam = ham = 'lunch'
```

spam ← 'lunch'

ham ← 'lunch'

```
spam += 42
```

spam ← spam + 42



- (1) Τι τιμή θα έχουν μετά την εκτέλεση της εντολής οι παρακάτω μεταβλητές;

```
>>> a, b, c, d = spam = foo, *boo = [10, 20, 'ham', 'eggs']
```

- Απαντήστε πρώτα χωρίς να εκτελέσετε την εντολή. Στη συνέχεια συγκρίνετε τα αποτελέσματα με την πρόβλεψή σας.
- (2) Ποιος θα ήταν ο απλούστερος κώδικας για αντιμετάθεση (swap) των τιμών 2 μεταβλητών a, b στην python;
  - Εξηγήστε χρησιμοποιώντας την αναφορά ονομάτων σε αντικείμενα-τιμές

# Δυναμική διαχείριση τύπων

Dynamic typing (“duck” typing)

Οι συναρτήσεις `type()` & `id()`

# Dynamic typing

## Δυναμική διαχείριση τύπων

- Η Python είναι γλώσσα όπου η διαχείριση τύπων δεδομένων γίνεται **δυναμικά**
- Χαρακτηρίζεται ως γλώσσα '**dynamically** typed' σε αντίθεση με 'statically typed' γλώσσες (πχ. C, Java)
- Αυτό ουσιαστικά σημαίνει πως **δεν** χρειάζεται να δηλωθεί ο τύπος μιας μεταβλητής εξ αρχής
- Αντίθετα ο τύπος διαμορφώνεται **δυναμικά** (δηλ. μπορεί να αλλάζει στην πορεία εκτέλεσης του προγράμματος) ανάλογα με το αντικείμενο στο οποίο αναφέρεται ένα όνομα μεταβλητής
- Περισσότερα: [Dynamic typing](#)

# Dynamic typing

Δυναμικές γλώσσες:

Η ταχύτητα - 1

- Οι δυναμικές γλώσσες είναι γενικά πιο **αργές** στην εκτέλεση του κώδικα από τις στατικές
- Αυτό οφείλεται κυρίως στο γεγονός πως στις στατικές γλώσσες πολλά θέματα διαχείρισης των δομών του κώδικα **καθορίζονται από την αρχή** στη φάση της μετάφρασης (compilation)
  - Ο compiler γνωρίζει πώς να βελτιώσει τον κώδικα καθώς γνωρίζει τον τύπο δεδομένων εξ αρχής
- Αντίθετα στις δυναμικές γλώσσες πολλά πράγματα πρέπει να ελέγχονται στη διάρκεια εκτέλεσης του κώδικα, κάτι που μειώνει την ταχύτητα εκτέλεσης

# Dynamic typing

Δυναμικές γλώσσες:

Η ταχύτητα - 2

- Για το θέμα της ταχύτητας:
- Α) Η διαφορά ταχύτητας αρχίζει και αποκτά σημασία κατά την επεξεργασία **μεγάλου πλήθους δεδομένων**, συνήθως σε επιστημονικού ενδιαφέροντος εφαρμογές και όχι στις περισσότερες άλλες εφαρμογές της γλώσσας
- Β) Υπάρχουν λύσεις που **αυξάνουν την ταχύτητα** εκτέλεσης κώδικα Python (δείτε επόμενη διαφάνεια)
- Γ) Η κοινή επαγγελματική πρακτική είναι πως γράφοντας μια εφαρμογή σε κώδικα Python (ή άλλη δυναμική γλώσσα) οι προγραμματιστές **βελτιώνουν** στα σημεία που χρειάζεται την ταχύτητα με κώδικα C, C++ ή ακόμα και Assembly
- Για περισσότερα δείτε:
  - [Why Python is Slow: Looking Under the Hood](#)
  - [Van Rossum: Python is not too slow](#)

# Dynamic typing

## Η λύση PyPy και άλλες...

- Υπάρχουν όμως διάφορες ουσιαστικές λύσεις για την **αύξηση της ταχύτητας εκτέλεσης** του κώδικα της δυναμικής CPython
- Πχ. JIT Compilers (Just In Time Compilation) που αυξάνουν δραματικά την ταχύτητα εκτέλεσης, όπως η PyPy
- Mypy
  - static type checker for Python
- Cython
  - a variant of Python that supports compilation to CPython C modules
- Nuitka
  - a static compiler that can translate Python programs to C++
- RPython
  - statically typed subsets of Python
- .....

# Dynamic typing

## Πλεονεκτεί σε κάτι η Python;

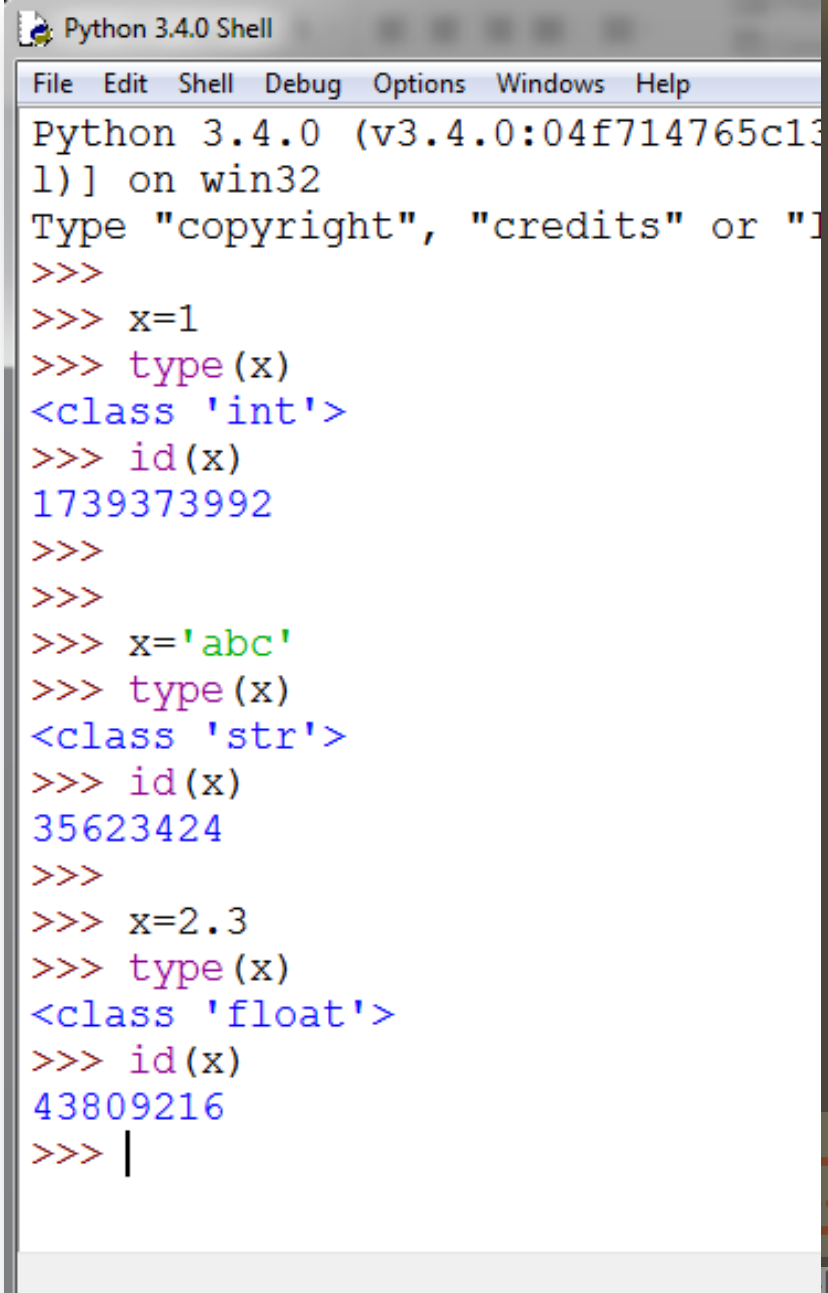
- Η Python είναι **ευκολότερη** στη χρήση από τις C γλώσσες και περισσότερο **ευέλικτη**
- Η **ευελιξία** οδηγεί σε βέλτιστη **αξιοποίηση** του χρόνου του προγραμματιστή για ανάπτυξη κώδικα
- Επίσης η Python προσφέρει εύκολη σύνδεση με πολλές **μεταφρασμένες βιβλιοθήκες** (compiled libraries)
- Έτσι:
- **Χρόνος ανάπτυξης (προγραμματιστή) ακριβότερος:** η Python επιτρέπει γρήγορη ανάπτυξη και μείωση αυτού του χρόνου
- **Πόροι hardware φθηνότεροι και γρηγορότεροι** (μνήμη, κλπ.): η Python αποτελεί καλή λύση όταν η διαχείριση των πόρων δεν είναι κρίσιμη ενώ προσφέρει πρακτικά αποδεκτό συμβιβασμό σε συνδυασμό με άλλες λύσεις (πχ. Compiled κώδικα όπου χρειάζεται υψηλή ταχύτητα, κλπ.)



# Οι συναρτήσεις **type()**, **id()**

- 1

- Η συνάρτηση **type()** επιστρέφει τον τύπο μιας μεταβλητής
- Χρησιμοποιήστε την **type()** για να προσδιορίσετε τον **τύπο της μεταβλητής** καθώς αυτός αλλάζει στη διάρκεια εκτέλεσης του κώδικα (πχ. παραδείγματα δεξιά)
- Χρησιμοποιήστε τη συνάρτηση **id()** για να δείτε τον μοναδιαίο **αριθμητικό αναγνωριστή (ταυτότητα)** του κάθε αντικειμένου στη μνήμη



```
Python 3.4.0 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.0 (v3.4.0:04f714765c13
1)] on win32
Type "copyright", "credits" or "
>>>
>>> x=1
>>> type(x)
<class 'int'>
>>> id(x)
1739373992
>>>
>>>
>>> x='abc'
>>> type(x)
<class 'str'>
>>> id(x)
35623424
>>>
>>> x=2.3
>>> type(x)
<class 'float'>
>>> id(x)
43809216
>>> |
```

# Οι συναρτήσεις **type()**, **id()**

- 2

- Γράψτε στον συντάκτη και τρέξτε τον κώδικα του παραδείγματος με την **type()** και την **id()**

```
x = input('Τιμή X: ')
print(x, type(x), ' ', id(x))

x = int(input('Τιμή X: '))
print(x, type(x), ' ', id(x))

x = float(input('Τιμή X: '))
print(x, type(x), ' ', id(x))
```

- Γιατί η **id(x)** δεν επιστρέφει το ίδιο αποτέλεσμα στον κώδικα δεξιά αφού καλείται για την ίδια μεταβλητή x;
- Σ' αυτήν και τις επόμενες ερωτήσεις απαντήστε χρησιμοποιώντας την αναφορά ονομάτων σε αντικείμενα-τιμές

```
>>> x = 1
>>> id(x)
1602600360
>>>
>>> x='spam'
>>> id(x)
44870432
>>>
```

# Ερώτηση

- 2

- Τι επιστρέφει η **type(x)** στον κώδικα δεξιά;
- Γιατί η **id(x)** και η πρώτη **id(y)** επιστρέφουν το ίδιο αποτέλεσμα, ενώ η δεύτερη κλήση της **id(y)** επιστρέφει διαφορετικό;

```
>>> x = 1
>>> type(x)
<class 'int'>
>>> id(x)
1602600360
>>>
>>> y = 1
>>> id(y)
1602600360
>>>
>>> y = 2
>>> id(y)
1602600376
>>>
```

# Ερώτηση

- 3

- Αν μετά την  $y=x$  κληθεί η **id(y)** τι κωδικό θα επιστρέψει;
- Αν κληθεί η **id(x)** τι κωδικό θα επιστρέψει;

```
>>> x=1
>>> id(x)
1602600360
>>>
>>> y=2
>>> id(y)
1602600376
>>>
>>> y=x
>>>
```

# Ερώτηση

- 4

- Τι θα επιστρέψει η τελευταία **type(1)** ;
- Γιατί;
- Παρόμοια, τι θα επιστρέψει η **type(0.5)**;

```
>>> x=1
>>> id(x)
1632943528
>>> type(x)
<class 'int'>
>>> type(1)
```

```
>>> x=0.5
>>> type(x)
<class 'float'>
>>> type(0.5)
```

- Τι θα επιστρέψει η **type(id(x))** στο παράδειγμα δεξιά;
- Στο ίδιο παράδειγμα τι θα επιστρέψει η **id(type(x))** ;

```
>>> x=1
>>>
>>> type(x)
<class 'int'>
>>>
>>> id(x)
1547943336
>>>
>>> type(id(x))
```

- Με βάση όλα τα προηγούμενα με ποια παραδείγματα και ποια επιχειρήματα θα εξηγήσετε την άποψη ότι «τα πάντα στην *Python* είναι αντικείμενα»;