



Dynamic Typing

Walter Cazzola

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
e-mail: cazzola@dico.unimi.it



Dynamic Typing Variables, Object and References

```
[22:55]cazzola@ulik:~/esercizi-pa>python3
>>> a = 42
```

As you know, Python is dynamically typed

- that is, there is no need to really explicit it.

Three separate concepts behind that assignment:

- **variable creation**, python works out names in spite of the (possible) content
- **variable types**, no type associated to the variable name, type lives with the object;
- **variable use** the name is replaced by the object when used in an expression

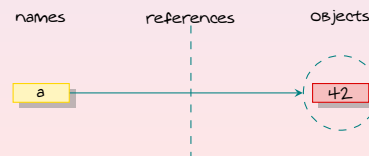


Dynamic Typing Variables, Object and References

```
[22:55]cazzola@ulik:~/esercizi-pa>python3
>>> a = 42
```

What happens inside?

1. create an object to represent the value 42;
 - **objects** are pieces of allocated memory;
2. create the variable a, if it does not exist yet;
 - **variable** are entries in a system table with spaces for links to objects;
3. link the variable a to the new object 42.
 - **references** are automatically followed pointers from variables to objects.



Dynamic Typing Types Live with Objects, Not Variables

```
[22:57]cazzola@ulik:~/esercizi-pa>python3
>>> a = 42           # it's an integer
>>> a = 'spam'       # now, it's a string
>>> a = 3.14         # now, it's a floating point
```

Coming from typed languages programming

- this looks as the type of a changes.

Of course, this is not true. In Python

names have no types

We simply changed the variable reference to a different object.

Objects know what type they are.

- Each object has an header field that tags it with its type.

Because objects know their type, variables don't have to.





```
[22:57]cazzola@ulik:~/esercizi-pa>python3
>>> a = 42
>>> a = 'spam'      # Reclaim 42 now (unless referenced elsewhere)
>>> a = 3.14        # Reclaim 'spam' now
>>> a = [1,2,3]     # Reclaim 3.14 now
```

- if it is not referenced by any other name or object



The diagram illustrates the relationship between variable names, references, and objects in Python. It is divided into four main sections:

- Code Block:** Contains the following Python code:


```
>>> a = 42
>>> b = a
>>> a = 'spam'
>>> b
>>> 42
```
- names:** Contains two yellow boxes labeled 'a' and 'B' (representing 'b').
- references:** A vertical dashed line separates the names from the objects. Arrows indicate the references:
 - A blue arrow from 'a' points to the '42' object.
 - A red arrow from 'B' points to the '42' object.
 - A blue arrow from 'a' points to the 'spam' object.
 - A blue arrow from 'B' points to the 'spam' object.
- objects:** Contains two dashed circles representing objects: one labeled '42' and one labeled 'spam'.

```
[23:00] cazzola@ulik:~/esercizi-pa>python3
>>> a = [1,2,3]
>>> b=a
>>> b[1]='spam'
>>> b
[1, 'spam', 3]
>>> a
[1, 'spam', 3]
```



```
[14:59]cazzola@ulik:~/esercizi-pa>python3
>>> L=[1,2,3]
>>> M=[1,2,3]
>>> N=L
>>> L==M, L is M
(True, False)
>>> L==N, L is N
(True, True)
```

```
>>> X=42
>>> Y=42
>>> X==Y,X is Y
(True, True)
```

```
>>> import sys
>>> sys.getrefcount(42)
10
>>> sys.getrefcount([1,2,3])
1
```



```
X = 42
L = [1,2,3]

def fake_mutable(i,l):
    i = i+2
    l[1] = '?!?!'
    l = {1,3,5,7}

[18:47] cazzola@ulik:~/esercizi-pa$ python3
>>> from args import fake_mutable, X, L
>>> print("X :- {0} \t L :- {1}".format(X,L))
X :- 42      L :- [1, 2, 3]
>>> fake_mutable(X,L)
>>> print("X :- {0} \t L :- {1}".format(X,L))
X :- 42      L :- [1, '?!?!', 3]
```

```
>>> L = [1,2,3]
>>> fake_mutable(X,L[:])
>>> print("X :- {0} \t L :- {1}".format(X,L))
X :- 42          L :- [1, 2, 3]
```

```
def mutable():
    global X, L
    X = X*2
    L[1] = '?!?!'
    L = {1,3,5,7}

if __name__ == "__main__":
    mutable()
    print("X :- {0} \t L :- {1}".format(X,L))
```



Closures in Action

Currying

Dynamic Typing

Walter Cazzola

Dynamic Typing

Definitions

surface

collection

equality

passing

arguments

Currying

References

$$f(x, y) = \frac{y}{x} \xrightarrow{f(2,3)} g(y) = f(2, y) = \frac{y}{2} \xrightarrow{g(3)} g(3) = \frac{3}{2}$$

```
def make_currying(f, a):
    def fc(*args):
        return f(a, *args)
    return fc

def f2(x, y):
    return x+y

def f3(x, y, z):
    return x+y+z

if __name__ == "__main__":
    a = make_currying(f2, 3)
    b = make_currying(f3, 4)
    c = make_currying(b, 7)
    print("(cf2 3){0}" .format(1,a(1),2,3,b(2,3)))
    print("((cf3 4) 7){0}" .format(5,c(5)))
```

```
[19:22]cazzola@ulic:~/esercizi-pa>python3 curry.py
(cf2 3)(1) :- 4, (cf3 4)(2,3) :- 9
((cf3 4) 7)(5) :- 16
```



Look at partial in functools.

Slide 9 of 10



References

Dynamic Typing

Walter Cazzola

Dynamic Typing

Definitions

surface

collection

equality

passing

arguments

Currying

References

- ▶ Jennifer Campbell, Paul Gries, Jason Montojo, and Greg Wilson.
Practical Programming: An Introduction to Computer Science Using Python.
The Pragmatic Bookshelf, second edition, 2009.
- ▶ Mark Lutz.
Learning Python.
O'Reilly, third edition, November 2001.
- ▶ Mark Pilgrim.
Dive into Python 3.
Apress*, 2009.



Slide 10 of 10