Python for Data Science

Short Introduction to Python

Dr. Ilkay Altintas

Python Introduction

C

```
#include "stdio.h"
int main() {
  printf("Hello\n");
}
```

Java

```
public class Hi {
  public static void main (String [] args) {
    System.out.println("Hello");
  }
}
```

C

```
#include "stdio.h"
int main() {
  printf("Hello\n");
}
```

python

```
print("hello")
```

Notice: no ;

Java

```
public class Hi {
  public static void main (String [] args) {
    System.out.println("Hello");
  }
}
```

C

```
#include "stdio.h"

int main() {
  int x = 3;
  int y = 4;
  printf("%s"\n,x+y);
}
```

python

```
x = 3
y = 4
print(x+y)
```

Notice: no types

Common Types in Python

- **Numeric:** integers, float, complex
- **Sequence:** list, tuple, range
- **Binary:** byte, bytearray
- **True/False:** bool
- **Text:** string

C

```
#include "stdio.h"


int main() {
  int x = 3;
  x = 4.5;
}
```

python

```
x = 3
x = 4.5
```

What happens when we run this in python?

C

```c
#include "stdio.h"


int main() {
  int x = 3;
  x = 4.5;
}
```

python

```python
x = 3
x = 4.5
```

Dynamic Typing!!

python

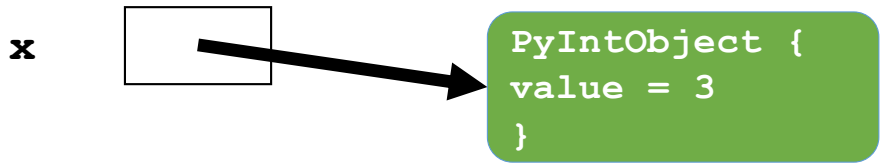
```
x = 3
```

```
PyIntObject {
   value;
   # other bookkeeping features
   # type, num_refs, etc.
}
```

python

```
x = 3
```

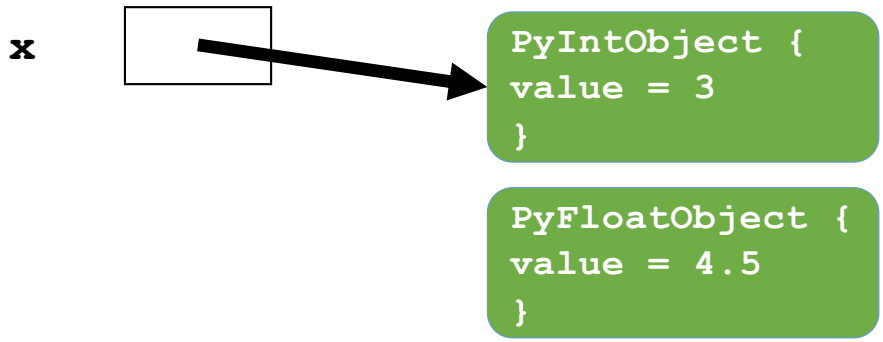
```
PyIntObject {
    value;
    # other bookkeeping features
    # type, num_refs, etc.
}
```



python

```
x = 3
x = 4.5
```

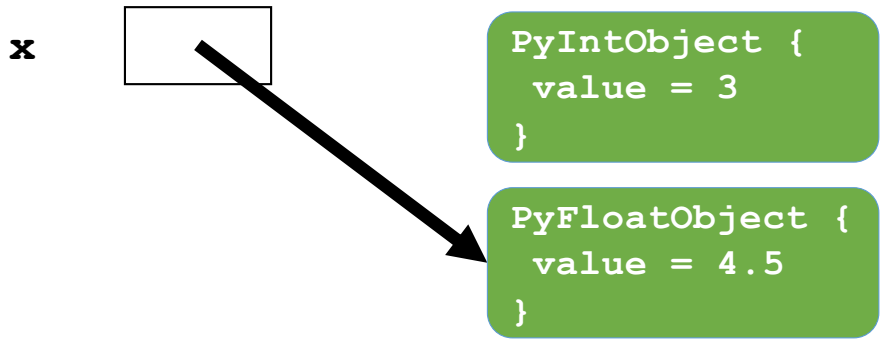
```
PyFloatObject {
    value;
    # other bookkeeping features
    # type, num_refs, etc.
}
```



python

```
x = 3
x = 4.5
```

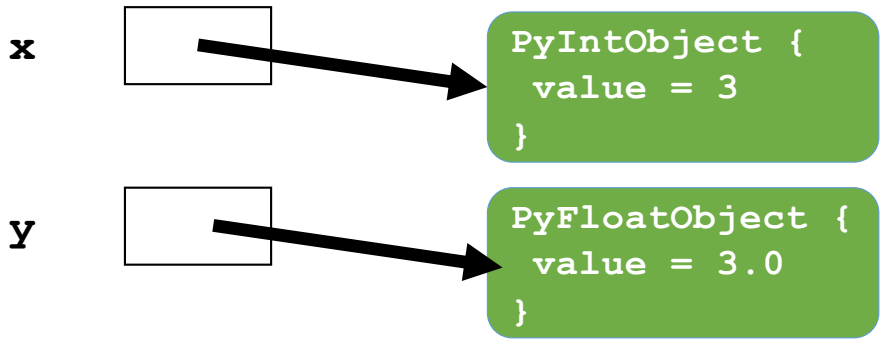
```
PyFloatObject {
    value;
    # other bookkeeping features
    # type, num_refs, etc.
}
```



Python shell

```
>>> x = 3
>>> y = 3.0
>>> x is y
```

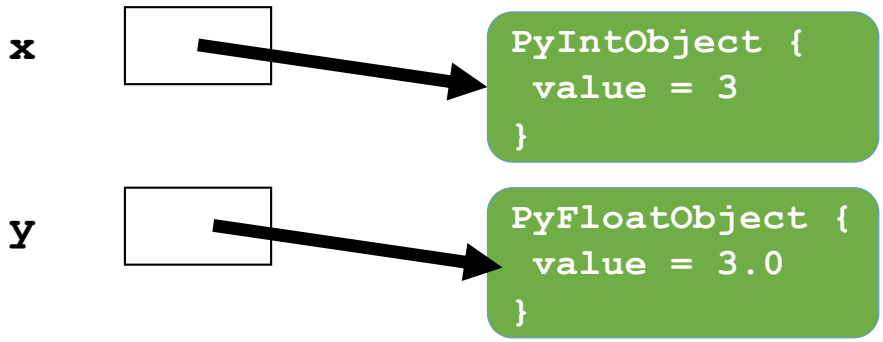
is returns if the references point to the same object



Python shell

```
>>> x = 3
>>> y = 3.0
>>> x is y
False
```

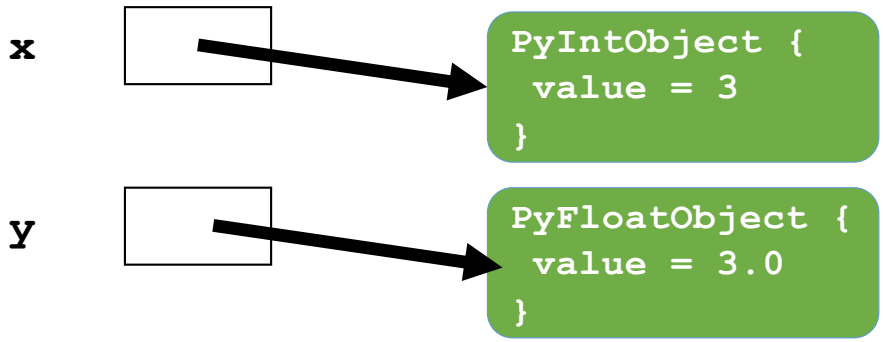
is returns if the references point to the same object



Python shell

```
>>> x = 3
>>> y = 3.0
>>> x == y
```

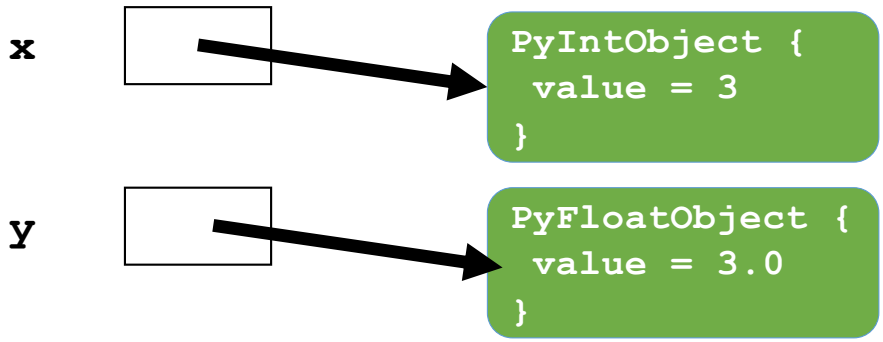
```
is returns if the references
point to the same object
== tests for equality
```



Python shell

```
>>> x = 3
>>> y = 3.0
>>> x == y
True
```

```
is returns if the references
point to the same object
== tests for equality
```



python

```
>>> x = "Hello"
>>> x.lower()
'hello'
```

<var_name>.<method_name>(params)

C

```c
#include "stdio.h"
int main() {
  int i = 0;
  for(i=0; i < 10; i++) {
    printf("%d\n",i);
  }
}
```

python

```python
for i in range(0,10):
        print(i)
```

Python uses indentation rather than brackets.

C

```
#include "stdio.h"
int main() {
  int i = 0;
  for(i=0; i < 10; i++) {
    printf("%d\n",i);
  }
}
```

python

```
for i in range(0,10):
        print(i)
```

```
range(start, stop[, step])
```

Returns values between **start** and **stop**, increasing by the value of **step** (defaults to 1).

python

```
for i in range(0,10,2):
        print(i)
```

What do you think this will print?

```
range(start, stop[, step])
```

Returns values between **start** and **stop**, increasing by the value of **step** (defaults to 1).

python

```
for i in range(0,10,2):
        print(i)
```

```
0
2
4
6
8
```

```
range(start, stop[, step])
```

Returns values between **start** and **stop**, increasing by the value of **step** (defaults to 1).

python

```
for i in range(0,10):
        ________________
                print(i)
```

```
0
2
4
6
8
```

What should I put here to print just the even values?

```
range(start, stop[, step])
```

Returns values between **start** and **stop**, increasing by the value of **step** (defaults to 1).

python

```
for i in range(0,10):

        ________________
                print(i)
```

```
0
2
4
6
8
```

What should I put here to print just the even values?

% (modulo)

x % y produces the remainder from x / y.

For example, 22%3 is 1 because 22 / 3 is 21 R1

python

```
for i in range(0,10):
        if i % 2 == 0:
                print(i)
```

```
0
2
4
6
8
```

What should I put here to print just the even values?

% (modulo)

x % y produces the remainder from x / y.

For example, 22%3 is 1 because 22 / 3 is 21 R1

python

```
for i in range(0,5):
        if i % 3 == 0:
                print(i)
        elif i % 3 == 1:
                print(i+10)
        else:
                print(i-10)
```

```
0
11
-8
3
14
```

% (modulo)

x % y produces the remainder from x / y.

For example, 22%3 is 1 because 22 / 3 is 21 R1

C

```c
int my_abs( int val) {
  if(val < 0) {
       return 0-val;
  }
  return val;
}
```

python

```python
def my_abs(val):
       if val < 0:
              return 0-val
       return val
```

python

```
def my_abs(val):
        if val < 0:
                return 0-val
        return val

print(my_abs(-7))
```

```
7
```

python

```
def my_abs(val):
        if val < 0:
                return 0-val
        return val

print(my_abs("Hi"))
```

```
Traceback (most recent call last):
  File "funct.py", line 6, in <module>
    print(my_abs("Hi"))
  File "funct.py", line 2, in my_abs
    if val < 0:
TypeError: unorderable types: str() < int()
```

List Basics

```
>>> list = [11,22,33]
>>> list
```

```
[11, 22, 33]
```

```
>>> list[1]
```

```
22
```

```
>>> list[3]
```

Error – index out of range

11	22	33
0	1	2

Iterating over a List

```
>>> list = [11,22,33]
>>> for i in list:
…         print(i)
11
22
33
```

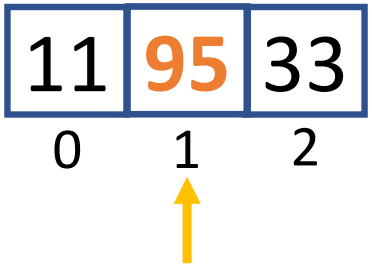
This loop is:

1. Easier to write
2. Less error prone
3. More readable

```
>>> for i in range(0,len(list)):
…         print(list[i])
11
22
33
```

Lists are MUTABLE

```
>>> list = [11,22,33]
>>> list[1]=95
```




```
>>> print(list[1])
```

```
95
```

Appending to a List

```
>>> list = [11,22,33]
```

```
>>> list.append(44)
```

```
>>> list
```

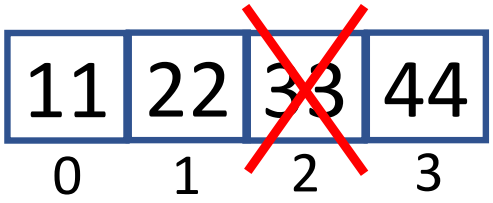
```
[11, 22, 33, 44]
```

11	22	33	44
0	1	2	3

Deleting from a List

```
>>> list = [11,22,33,44]
>>> list.pop(2)
```

```
33
```



Tuples Basics

'Honda'	'Civic'	4	2017
0	1	2	3

```
>>> tuple1 = ('Honda','Civic',4,2017)
>>> tuple1
```

```
('Honda', 'Civic', 4, 2017)
```

```
>>> tuple1[1]
```

```
'Civic'
```

```
>>> len(tuple1)
```

```
4
```

Tuples are IMMUTABLE

```
>>> tuple1 = ('Honda','Civic',4,2017)
>>> tuple[3]=2018
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Immutability Matters

If an object is immutable, you can TRUST it to never change!

What are Dictionaries

Key	Value
'A12367'	'David Wu'
'A27691'	'Maria Sanchez'
'A16947'	'Tim Williams'
'A21934'	'Sarah Jones'

Dictionary Examples

Key	Value
'CSE8A'	['Christine Alvarado', 'Beth Simon', 'Paul Cao']
'CSE141'	['Dean Tullsen', 'Steve Swanson', 'Leo Porter']
…	…

Dictionary Examples

Movie	Rating
Ghostbusters 2016	
Ghostbusters 1984	

http://www.imdb.com/

Movie Ratings Dictionary

Key	Value
('Ghostbusters',2016)	5.4
('Ghostbusters',1984)	7.8
('Cars',2006)	7.1
…	…

http://www.imdb.com/

Dictionary Basics

```
>>> dict = {('Ghostbusters',2016): 5.4,
('Ghostbusters',1984):7.8}
>>> tuple1
```

```
{('Ghostbusters', 2016): 5.4,
('Ghostbusters', 1984): 7.8}
```

```
>>>  dict[('Ghostbusters',2016)]
```

```
5.4
```

```
>>> len(dict)
```

```
2
```

Adding to a Dictionary

```
>>> dict = {('Ghostbusters',2016): 5.4,
('Ghostbusters',1984):7.8}
>>> dict[('Cars',2006)] = 7.1
```

```
>>>  dict
```

```
{('Ghostbusters', 2016): 5.4,
('Cars', 2006): 7.1,
('Ghostbusters', 1984): 7.8}
```

Dictionaries are unordered

Getting a value from a dictionary

```
>>> dict = {('Ghostbusters', 2016): 5.4,
('Ghostbusters', 1984): 7.8, ('Cars', 2006):7.1}
>>> x = dict[('Cars',2006)]
>>> x
```

```
7.1
```

```
>>> x = dict[('Toy Story',1995)]
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
KeyError: ('Cars', 2000)
```

Safer way to get from a dictionary

```
>>> dict = {('Ghostbusters', 2016): 5.4,
('Ghostbusters', 1984): 7.8, ('Cars', 2006):7.1}
>>> x = dict.get(('Cars',2006))
>>> x
```

```
7.1
```

```
>>> x = dict.get(('Toy Story',1995))
>>> x == None
```

```
True
```

```
>>> ('Toy Story',1995) in dict
```

```
False
```

Deleting from a Dictionary

```
>>> dict = {('Ghostbusters', 2016): 5.4,
('Ghostbusters', 1984): 7.8, ('Cars', 2006):7.1}
>>> dict.pop(('Ghostbusters',2016))
```

```
5.4
```

```
>>>  dict
```

```
{('Cars', 2006): 7.1,
('Ghostbusters', 1984): 7.8}
```

```
>>> del dict[('Cars', 2006)]
```

Iterating over a dictionary

```
>>>  dict = {('Ghostbusters', 2016): 5.4,
('Ghostbusters', 1984): 7.8, ('Cars', 2006):7.1}
>>> for i in dict:
…         print(i)
```

```
('Ghostbusters', 2016)
('Cars', 2006)
('Ghostbusters', 1984)
```

Iterating over a dictionary

```
>>>  dict = {('Ghostbusters', 2016): 5.4,
('Ghostbusters', 1984): 7.8, ('Cars', 2006):7.1}
>>> for key, value in dict.items():
…         print(key,":",value)
```

```
('Ghostbusters', 2016) : 5.4
('Cars', 2006) : 2.1
('Ghostbusters', 1984) : 7.8
```

Be CAREFUL while iterating

```
>>>  dict = {('Ghostbusters', 2016): 5.4,
('Ghostbusters', 1984): 7.8, ('Cars', 2006):7.1}
>>> for i in dict:
…         dict.pop(i)
```

```
5.4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: dictionary changed size
during iteration
```

Selective removal

```
>>>  dict = {('Ghostbusters', 2016): 5.4,
('Ghostbusters', 1984): 7.8, ('Cars',
2006):7.1}
>>> to_remove = [];
>>> for i in dict:
…         if(i[1] < 2000):
…             to_remove.append(i)
>>> for i in to_remove:
...     dict.pop(i)
...
>>> dict
{('Ghostbusters', 2016): 5.4,
('Cars', 2006): 2.1}
```