

# A Short and Incomplete Introduction to Python

## Part 1: Basic Python syntax

**Riccardo Murri** <riccardo.murri@uzh.ch>,  
Sergio Maffioletti <sergio.maffioletti@uzh.ch>  
S3IT: Services and Support for Science IT,  
University of Zurich

# **Python basics**

## Lines of Python code

Line of Python code are ended by the “new line” character. (I.e., when you press the *Enter* key.)

A line can be continued onto the next by ending it with the character ‘\’; for example:

```
In [1]: "hello" + \
...: " world!"
```

**Out [1]:** 'hello world!'

The prompt changes to ‘...’ on continuation lines.

Reference:

[http://docs.python.org/reference/lexical\\_analysis.html#line-structure](http://docs.python.org/reference/lexical_analysis.html#line-structure)

## String literals, I

There are several ways to express string literals in Python.

Single and double quotes can be used interchangeably:

```
In [2]: "a string" == 'a string'  
Out[2]: True
```

You can use the single quotes inside double-quoted strings, and viceversa:

```
In [3]: a = "Isn't it ok?"  
In [4]: b = '"Yes", he said.'
```

## String literals, II

Multi-line strings are delimited by three quote characters.

```
In [5]: a = """This is a string,  
....: that extends over more  
....: than one line.  
....: """
```

In other words, you need not use the backslashes “\” at the end of the lines.

# Operators

All the usual unary and binary arithmetic operators are defined in Python: `+`, `-`, `*`, `/`, `**` (exponentiation), `<<`, `>>`, etc.

Logical operators are expressed using plain English words: `and`, `or`, `not`.

Numerical and string comparison also follows the usual notation: `<`, `>`, `<=`, `==`, `!=`, ...

## *Reference:*

- ▶ <http://docs.python.org/library/stdtypes.html#boolean-operations-and-or-not>
- ▶ <http://docs.python.org/library/stdtypes.html#comparisons>

## Your first exercise

How much is  $2^{144}$  ?

(You have 1 minute time.)

## Operators, II

Some operators are defined for non-numeric types:

```
>>> "Py" + 'thon'  
'Python'
```

Some support operands of mixed type:

```
>>> "a" * 2  
'aa'  
>>> 2 * "a"  
'aa'
```

Some do not:

```
>>> "aaa" / 3  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

## Operators, III

The “%” operator computes the remainder of integer division.

In [6]: 9 % 2

Out [6]: 1

It also doubles up as *string interpolation operator*, but the ‘`format()`’ method (see next slide) is more convenient.

## String interpolation

The `.format()` method can be used to substitute values into placeholder strings.

Placeholders can indicate substitutions by ordinal number:

```
>>> "This is slide {0} of {1} ".format(20, 1001)
'This is slide 20 of 1001.'
```

You can use names instead of numbers (then the order parameter occur in `format()` does not matter):

```
>>> "Today is {month} {day} ".format(day=2, month='March')
'Today is March 2'
```

Reference: <https://pyformat.info/>

## String interpolation

The `.format()` method can be used to substitute values into placeholder strings.

Placeholders can indicate substitutions by ordinal number:

```
>>> "This is slide {0} of {1}" .format(20, 1001)
'This is slide 20 of 1001.'
```

You can use names instead of numbers (then the order parameter occur in `format()` does not matter):

```
>>> "Today is {month} {day}" .format(day=2, month='March')
'Today is March 2'
```

Reference: <https://pyformat.info/>

## String interpolation

The `.format()` method can be used to substitute values into placeholder strings.

Placeholders can indicate substitutions by ordinal number:

```
>>> "This is slide {0} of {1}" .format(20, 1001)
'This is slide 20 of 1001.'
```

You can use names instead of numbers (then the order parameter occur in `format()` does not matter):

```
>>> "Today is {month} {day}" .format(day=2, month='March')
'Today is March 2'
```

Reference: <https://pyformat.info/>

# Assignment, I

Assignment is done via the '=' statement:

**In [7]:** `a = 1`

**In [8]:** `print(a)`

**Out [8]:** 1

There are a few shortcut notations:

`a += b` is short for `a = a + b`,

`a -= b` is short for `a = a - b`,

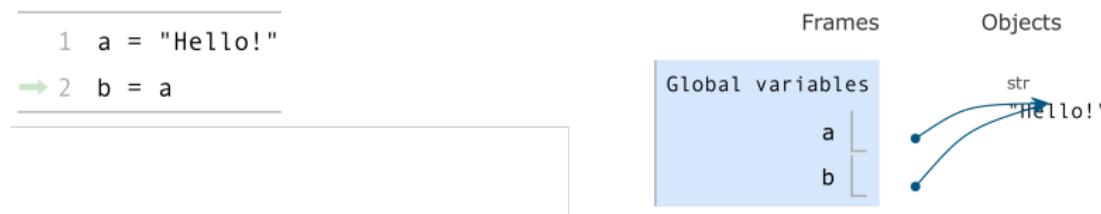
`a *= b` is short for `a = a * b`,

etc. — one for every legal operator.

## Assignment, II

**Python variables are just “names” given to values.**

This allows you to *reference* the string ‘Python’ by the *name* a. But also by another name b:



The *same* object can be given many names!

*See also:* <http://excess.org/article/2014/04/bar-foo/>

## The **is** operator

The **is** operator allows you to test whether two names refer to the same object:

```
>>> a = 1  
>>> b = 1  
>>> a is b  
True
```

## Basic types

Basic object types in Python 3:

`bool` The class of the two boolean constants  
True, False.

`int` Integer numbers: 1, -2, ...

`float` Double precision floating-point numbers,  
e.g.: 3.1415, -1e-3.

`bytes` String of byte-size characters.

`str` Text (string of UNICODE characters).

`list` Mutable list of Python objects

`dict` Key/value mapping

The type of a Python object can be gotten via the `type()` function:

**In [3]:** `type('hello')`

**Out[3]:** str

# **Appendix**

## All variables are references

In Python, **all objects are ever passed by reference**.

In particular, **variables always store a reference to an object**, never a copy!

Hence, you have to be careful when modifying objects:

```
>>> a = [1, 2, 3]
>>> b = a
>>> b.remove(2)
>>> print(a)
???
```

**Q:** How many items are in the a list now?

## All variables are references

In Python, **all objects are ever passed by reference**.

In particular, **variables always store a reference to an object**, never a copy!

Hence, you have to be careful when modifying objects:

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

Run this example in the [Online Python Tutor](#) to better understand what's going on.

*This applies particularly for variables that capture the arguments to a function call!*

# All variables are references (demo)

[www.pythontutor.com](http://www.pythontutor.com)

```
→ 1 a = [1, 2, 3]
  2 b = a
  3 b.remove(2)
  4 print a
  5 print b
```

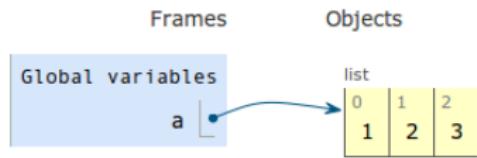
Frames

Objects

## All variables are references (demo)

[www.pythontutor.com](http://www.pythontutor.com)

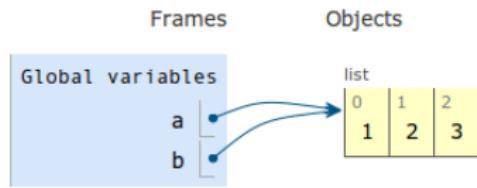
```
→ 1 a = [1, 2, 3]
→ 2 b = a
  3 b.remove(2)
  4 print a
  5 print b
```



## All variables are references (demo)

[www.pythontutor.com](http://www.pythontutor.com)

```
1 a = [1, 2, 3]
→ 2 b = a
→ 3 b.remove(2)
4 print a
5 print b
```



# All variables are references (demo)

[www.pythontutor.com](http://www.pythontutor.com)

---

```
1 a = [1, 2, 3]
2 b = a
3 b.remove(2)
4 print a
5 print b
```

---

