UNIVERSITY OF PORTSMOUTH

Python for Data Analysis
Python Basics (Week 2)

Atefeh Khazaei

atefeh.khazaei@port.ac.uk

What we will learn this week?

- Python Language Semantics
- Python Scalar Types
- Control Flow in Python

Language Semantics
Indentation, not braces

- Python uses whitespace (tabs or spaces) to structure code instead of using braces as in many other languages like R, C++, Java, and Perl.

```
if age >= 18:
print("You can vote")
else:
print("Sorry, you can't vote")
```

```
  File "<ipython-input-1-ea1a652de2fa>", line 2
    print("You can vote")
    ^
IndentationError: expected an indented block
```

```
if age >= 18:
    print("You can vote")
else:
    print("Sorry, you can't vote")
```

Language Semantics (cont.)
Everything is an object

- An important characteristic of the Python language is the consistency of its object model.
- Every number, string, data structure, function, class, module, and so on exists in the Python interpreter in its own “box,” which is referred to as a Python object.
- In practice, this makes the language very flexible, as even functions can be treated like any other object.

Language Semantics (cont.)
Attributes and methods

- Objects in Python typically have both **attributes** (other Python objects stored "inside" the object) and **methods** (functions associated with an object that can have access to the object's internal data).
- Both of them are accessed via the syntax **obj.attribute_name**

```
import numpy
```

```
data = numpy.random.randn(2, 3)
```

```
data
```

```
array([[-1.87518596, -0.2064468 , -0.68615051],
       [ 2.01444836,  0.85566718, -0.89741156]])
```

```
In [1]: a = 'test'
```

```
In [ ]: # a.<Press Tab>
        a.
```

```
capitalize
casefold
center
count
encode
endswith
expandtabs
find
format
format_map
```

Language Semantics (cont.)
Comments

- Any text preceded by the hash mark # is ignored by the Python interpreter.
- This is often used to add comments to code.

```
# This is a comment
print("Hello World!")
```

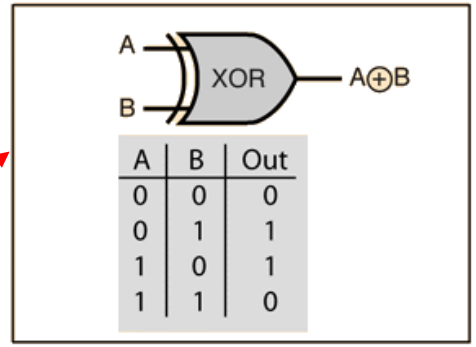
```
Hello World!
```

Language Semantics (cont.)

Binary operators and comparisons

- Most of the binary math operations and comparisons are as you might expect.

Operation	Description
`a + b`	Add a and b
`a - b`	Subtract b from a
`a * b`	Multiply a by b
`a / b`	Divide a by b
`a // b`	Floor-divide a by b, dropping any fractional remainder
`a ** b`	Raise a to the b power
`a & b`	`True` if both a and b are `True`; for integers, take the bitwise AND
`a | b`	`True` if either a or b is `True`; for integers, take the bitwise OR
`a ^ b`	For booleans, `True` if a or b is `True`, but not both; for integers, take the bitwise `EXCLUSIVE-OR`
`a == b`	`True` if a equals b
`a != b`	`True` if a is not equal to b
`a <= b, a < b`	`True` if a is less than (less than or equal) to b
`a > b, a >= b`	`True` if a is greater than (greater than or equal) to b
`a is b`	`True` if a and b reference the same Python object
`a is not b`	`True` if a and b reference different Python objects



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Language Semantics (cont.)

Binary operators and comparisons

- Simple examples:

```
a = 5
b = 3
a / b
```

```
1.6666666666666667
```

```
a // b
```

```
1
```

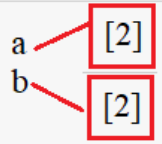
```
a % b
```

```
2
```

```
a ** b
```

```
125
```

```
a = [2]
b = [2]
a == b
```




```
True
```

```
a is b
```

```
False
```

```
a = [2]
b = a
a == b
```


```
True
```

```
a is b
```

```
True
```

Scalar Types

Standard Python scalar types

Type	Description
`None`	The Python "null" value (only one instance of the `None` object exists)
`str`	String type; holds Unicode (UTF-8 encoded) strings
`bytes`	Raw ASCII bytes (or Unicode encoded as bytes)
`float`	Double-precision (64-bit) floating-point number (note there is no separate `double` type)
`bool`	A `True` or `False` value
`int`	Arbitrary precision signed integer

Numeric types

Scalar Types (cont.)
Type casting

- The *str*, *bool*, *int*, and *float* types are also functions that can be used to cast values to those types:

```
kilos = float(input("Enter a weight in kilos: "))
pounds = 2.2 * kilos
print("The weight in pounds is", pounds)
```

Scalar Types (cont.)
Bytes and Unicode

- In modern Python (i.e., Python 3.0 and up), Unicode has become the first-class string type to enable more consistent handling of ASCII and non-ASCII text.
  - Unicode, formally the Unicode Standard, is an information technology standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems.
  - Useful feature for text analysis using python
- In older versions of Python, strings were all bytes without any explicit Unicode encoding.

Scalar Types (cont.)

None

- None is the Python null value type.
- If a function does not explicitly return a value, it implicitly returns None.
- None is also a common default value for function arguments.

```
def add_and_maybe_multiply(a, b, c=None):
    result = a + b

    if c is not None:
        result = result * c

    return result
```

Scalar Types (cont.)
Dates and times

- The built-in Python **datetime** module provides datetime, date, and time types.

```
In [1]: from datetime import datetime, date, time
        dt = datetime(2021, 10, 20, 9, 30, 21)

In [2]: dt.day

Out[2]: 20

In [3]: dt.minute

Out[3]: 30
```

Control Flow
if, elif, and else

- The ***if*** statement is one of the most well-known control flow statements.
- An if statement can be optionally followed by one or more ***elif*** blocks and a catchall ***else*** block if all of the conditions are False:

```
if x < 0:
    print('It's negative')
elif x == 0:
    print('Equal to zero')
elif 0 < x < 5:
    print('Positive but smaller than 5')
else:
    print('Positive and larger than or equal to 5')
```

Control Flow (cont.)
for loops

- ***for*** loops are for iterating over a collection (like a list or tuple) or an iterator.

  The standard syntax for a for loop is:

```
for value in collection:
    # do something with value
```

- For example:

More information and examples in practical sheet

```
for i in range(4):
    for j in range(4):
        if j > i:
            break
        print((i, j))
```

Control Flow (cont.)
while loops

- A ***while*** loop specifies a condition and a block of code that is to be executed until the condition evaluates to False or the loop is explicitly ended with *break*:

```
x = 256
total = 0
while x > 0:
    if total > 500:
        break
    total += x
    x = x // 2
```

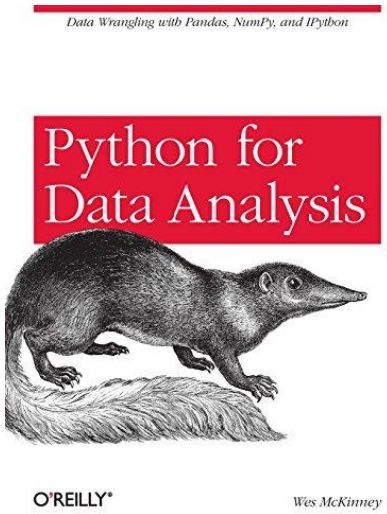
Control Flow (cont.)

pass

- pass is the “no-op” statement in Python. It can be used in blocks where no action is to be taken (or as a placeholder for code not yet implemented); it is only required because Python uses whitespace to delimit blocks:

```
if x < 0:
    print('negative!')
elif x == 0:
    # TODO: put something smart here
    pass
else:
    print('positive!')
```

References & More Resources



- References:
  - McKinney, Wes. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc., 2012.

- More Resources:
  - Python Data Analysis on Linkedin Learning:

    https://www.linkedin.com/learning/python-data-analysis-2
  - Learning Python on Linkedin Learning

    https://www.linkedin.com/learning/learning-python
    - To use Linkedin Learning, you can log in with your university account:

      https://myport.port.ac.uk/study-skills/linkedin-learning

COURSE

Python Data Analysis

By: Michele Vallisneri

COURSE

Learning Python

By: Joe Marini

Practical Session

- Please read the practical sheet (Week02_Practicals.pdf) carefully.
- The following items are included in this file:
  - Supplementary lecture tips
  - Various examples
  - Various exercises
- Do the exercises.