

Programming in python™ for Business Analytics (BMAN73701)

Week 1/ Lecture 1 – Introduction

Dr. Fanlin Meng

fanlin.meng@manchester.ac.uk

Office: AMBS 3.094

Virtual office: <https://zoom.us/j/94473763354>

Office Hours: Monday 12-1pm

Who am I?

- **Name:** Fanlin Meng
- **Position:** Lecturer in Data Science
- **Education:**
 - BSc/ MSc in Automation
 - PhD in Computer Science
- **Research interests:** Machine learning, optimization, game theory with applications in energy market, smart energy, transport, etc.

Who is in the room?

- MSc Business Analytics
- MSc Data Science
- MSc students from other programs
- MSc and PhD students auditing the module

Outline for today

- **General info about the course**
- Whetting your appetite
- Setting up your Integrated Analysis Environment
- Python essentials
 - Your first program
 - Using Python as a Calculator
 - Numeric data types
 - String operations
 - Type conversion
 - Variables

Previous programming experience ...

A. None



B. Intermediate



Programming: A series of instructions that will automate performing a specific task of solving a given problem.

C. Expert



Why Programming is True Myths?

Good coders work
long hours

Good programmers
write their most
important code in the
shower and in their
dreams

Good programmers
are less likely to be
randomly looking for
work

Counting starts from
zero, not one

Programming is
thinking, not typing

I can do

Genius (With
An IQ of 160)

Future prospects!?



Digital Marketing

Strategies Focused on Increasing The Reach & Visibility of Your Business.



Quantitative
vs
Qualitative

© 2013 Barnett, Jack

Course aims

- An **introduction** to the fundamentals of **Python**, a general-purpose programming language
- **Application of Python** to Data Science, Big Data Analytics and Optimization to business problems
- Use Python to **facilitate decision making**

```
class Person:

    population = 0

    def __init__(self, name, age):
        self.name = name
        self.age = age
        print(self.name)
        print(self.age)
        Person.population += 1
        print("population: ", Person.)

    def returnName(self):
        print("name: ", self.name)

    def setName(self, newName):
        print(self.name, " is now ", newName)
        self.name = newName

    def returnAge(self):
        print("age: ", self.age)

    def addAge(self):
        self.age += 1
        print(self.name, " is now ", self.age)
```



Learning outcomes

- At the end of the course, you should be able to:
 - Read and write Python code and understand how to use Python packages.
 - Implement algorithms of moderate complexity in Python.
 - Understand the fundamentals of object-oriented programming using Python.
 - Understand how to implement simple data science and optimization algorithms from the literature to tackle business applications.
 - Develop your own algorithms to solve basic data science and optimization problems.
 - Use Python packages to solve complex data science, visualisation and optimisation problems in business and management (e.g., portfolio optimization, customer segmentation, and analysis of financial data).

Course assessment

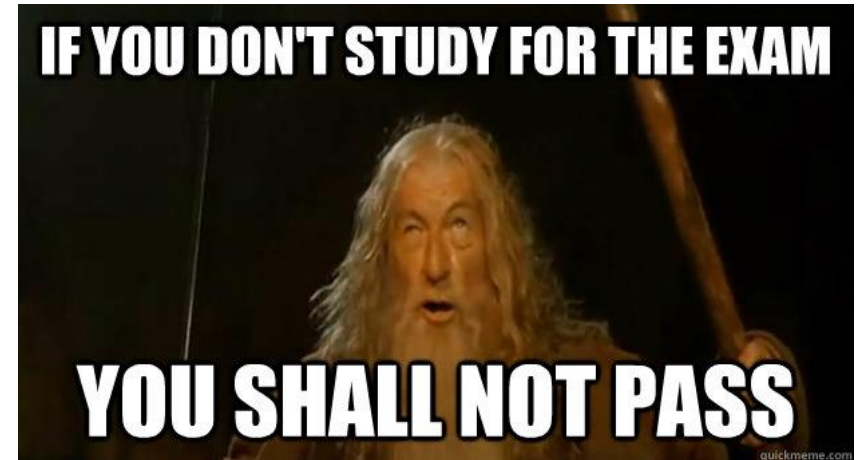
- **Formative assessment:**

- Online quizzes
- Seminars

- **Summative assessment:**

70% coursework and 30% online tests

- **Coursework** due in Week 11 (8th Dec, Friday, 3pm)
 - **Group report (35%):** 5000 words + executable Python script
 - **Group presentation (25%):** 10min
 - **Peer assessment (10%)**
- **Online tests**
 - 2 online tests (15% each) in Week 7 and 10





Fanlin Meng

Week	Topic
1	Course Intro + Setting up your integrated analysis environment
1	Conditionals and loops
2	Function, modules and exceptions
2	Object-oriented programming I
3	Shallow/deep copy, reading/writing to files, functions with variable arguments
3	Numerical analysis
4	Data exploration and visualization
4	Data processing and preparation
5	Introduction to machine learning in Python I
5	Introduction to machine learning in Python II algorithms II
TBC	Revision lecture potentially

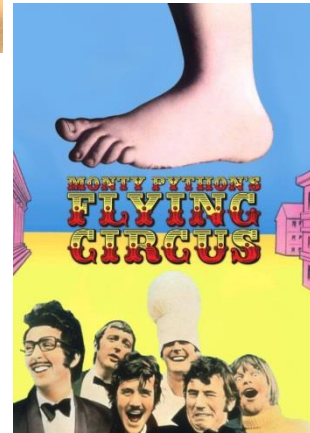
Xian Yang

Outline for today

- General info about the course
- **Whetting your appetite**
- Setting up your Integrated Analysis Environment
- Python essentials
 - Your first program
 - Using Python as a Calculator
 - Numeric data types
 - String operations
 - Type conversion
 - Variables

History of Python

- First appeared in 1991 thanks to **Guido van Rossum**
- **Google** (2005 – Dec 2012), **Dropbox** (2013 - 2019), **Microsoft** (since 2020)
- *“I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of **Monty Python's Flying Circus**).”* (Guido van Rossum, 1996)



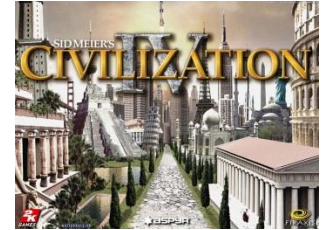
Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.

Why Python?

- Used by e.g.



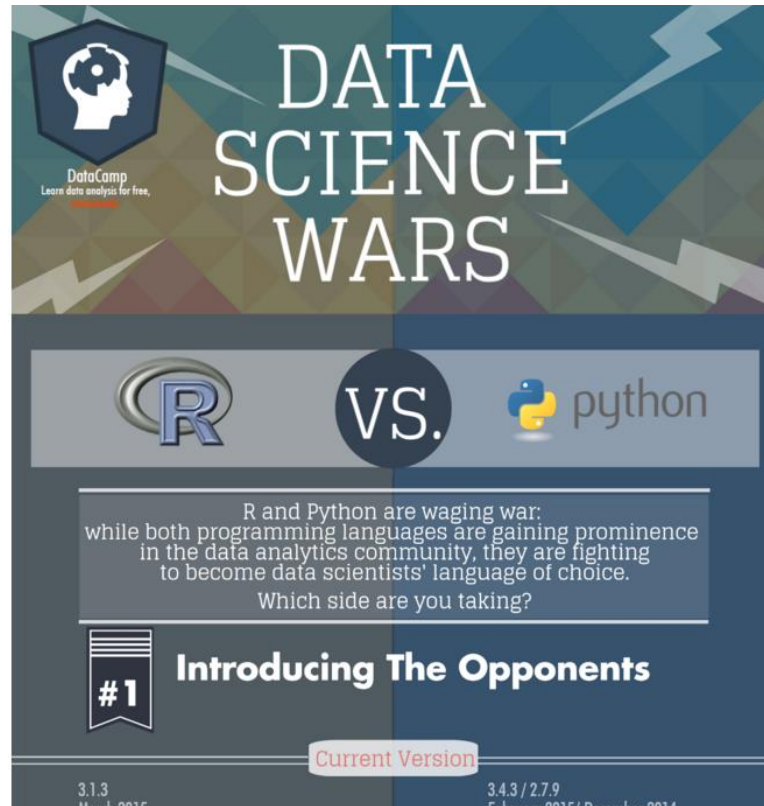
ABN·AMRO



- Very attractive for **Agile Software Development**
- Very attractive for use as a **scripting language** (similar to Unix shell script or Windows batch files) but much more powerful! → applicable for GUI applications, games, and more
- **Open-source, simple and easy to learn syntax, short codes**
- **No compilation step** → fast edit-test-debug cycle → increased productivity
- **Debugging is easy** → error raises an exception
- **Huge user community** leading to loads of supporting online material, tutorials, discussion groups, blogs, books, etc

Comparing Python to other languages

- <https://www.python.org/doc/essays/comparisons/>



- and many more comparisons available online...

Supporting online information

- Python manual <https://www.python.org/doc/>
- Learn Python <http://www.learnpython.org/>
- Learn Python – mobile phone app
https://play.google.com/store/apps/details?id=com.sololearn.python&hl=en_GB
- IPython <http://ipython.readthedocs.io/en/stable/index.html>
- Python tutorial <https://docs.python.org/3/tutorial/>
- Many books, such as [Think Python](#) and others (see reading list on course outline)
- <http://www.tutorialspoint.com/codingground.htm>

Assumption is we are using Python 3.5 or later (<https://docs.python.org/3.5/contents.html>)

Outline for today

- General info about the course
- Whetting your appetite
- **Setting up your Integrated Analysis Environment**
- Python essentials
 - Your first program
 - Using Python as a Calculator
 - Numeric data types
 - String operations
 - Type conversion
 - Variables

Installing Python

Windows:

- Download Python from <http://www.python.org>
- Install Python
- Run **Idle** from the Start Menu

Mac OS X:

- Python is already installed.
- Open a terminal and run `python` or run Idle from Finder.

Linux:

- Chances are you already have Python installed. To check, run `python` from the terminal.
- If not, install from your distribution's package system.

<https://www.python.org/downloads/>

Programming environment

- **Anaconda:** Freemium open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment.
- **Spyder:** Open source cross-platform IDLE (Integrated DeveLopment Environment) for scientific programming in the Python language. Spyder integrates NumPy, SciPy, Matplotlib and IPython, as well as other open source software.



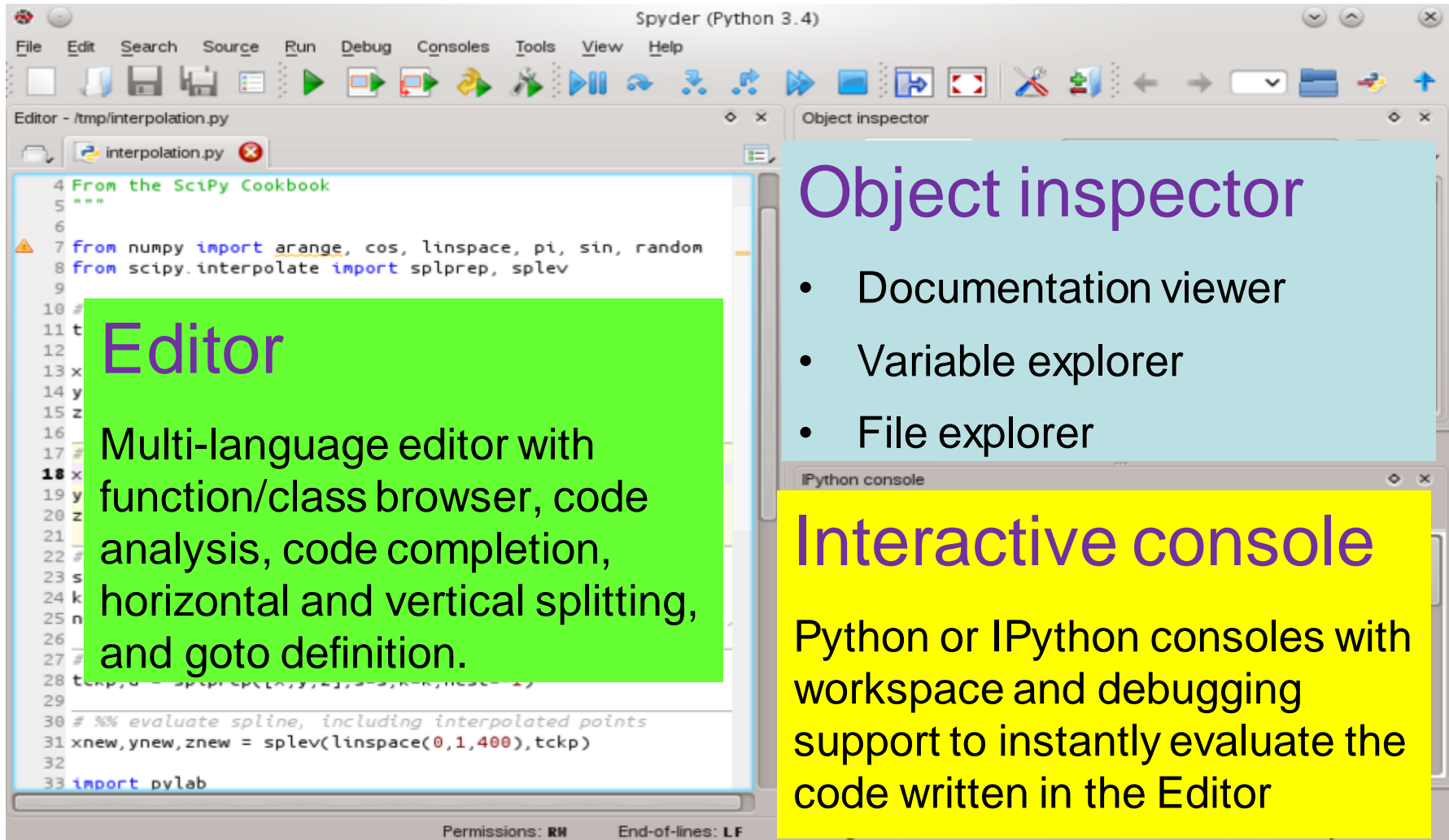
More than 30 other editors available...

<https://wiki.python.org/moin/PythonEditors>



Spyder: Features

Launch: Run spyder.exe from Start Menu



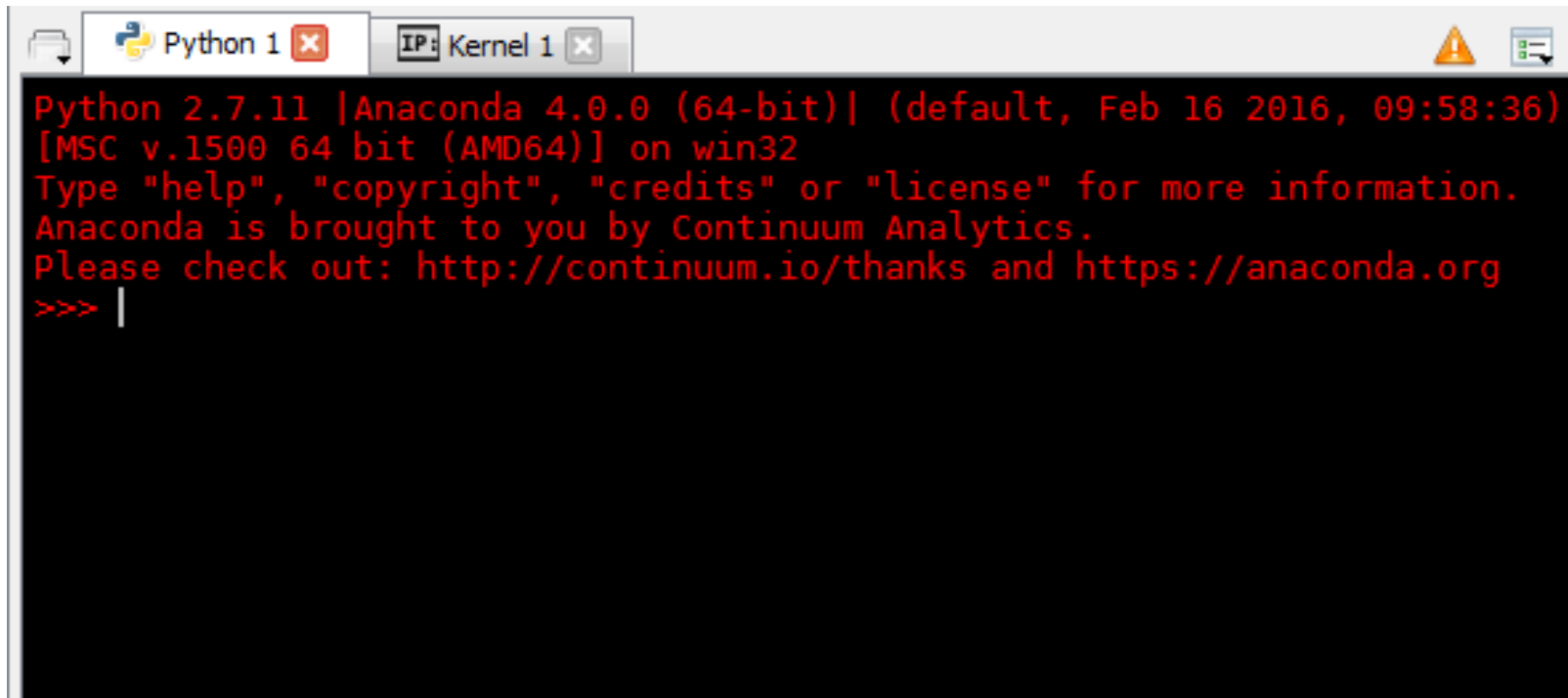
The screenshot displays the Spyder Python IDE interface. The main window is titled "Spyder (Python 3.4)". It features a menu bar with options: File, Edit, Search, Source, Run, Debug, Consoles, Tools, View, and Help. Below the menu bar is a toolbar with various icons for file operations, running code, and debugging. The interface is divided into several panes:

- Editor:** The central pane shows a Python script named "interpolation.py". The code includes comments and imports from numpy and scipy. A green box highlights the text: "Editor: Multi-language editor with function/class browser, code analysis, code completion, horizontal and vertical splitting, and goto definition."
- Object inspector:** A pane on the right side, titled "Object inspector", which is currently empty. A light blue box highlights the text: "Object inspector: Documentation viewer, Variable explorer, File explorer".
- Python console:** A pane at the bottom right, titled "Python console", which is currently empty. A yellow box highlights the text: "Interactive console: Python or IPython consoles with workspace and debugging support to instantly evaluate the code written in the Editor".

At the bottom of the window, there are status bars showing "Permissions: RW" and "End-of-lines: LF".

Python interpreter

- Text can be typed whenever the >>> sign is shown (prompt)
- Write commands one-at-a-time and see results immediately



The screenshot shows a Jupyter Notebook window with a tab labeled 'Python 1'. The interface includes a toolbar with icons for file operations and a warning icon. The main area is a black terminal window with red text. The text displays the Python version (2.7.11), the Anaconda environment (4.0.0, 64-bit), and the system (MSC v.1500 64 bit (AMD64) on win32). It also provides instructions on how to get help, copyright, credits, or license information, and mentions that Anaconda is brought to you by Continuum Analytics. The prompt '>>>' is shown at the end of the line, with a cursor following it.

```
Python 2.7.11 |Anaconda 4.0.0 (64-bit)| (default, Feb 16 2016, 09:58:36)  
[MSC v.1500 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
Anaconda is brought to you by Continuum Analytics.  
Please check out: http://continuum.io/thanks and https://anaconda.org  
>>> |
```

Outline for today

- General info about the course
- Whetting your appetite
- Setting up your Integrated Analysis Environment
- **Python essentials**
 - Your first program
 - Using Python as a Calculator
 - Numeric data types
 - String operations
 - Type conversion
 - Variables

Your first program

Type in “print(“Hello world!”)” and press enter. You should see the following

```
>>> print("Hello world!")  
Hello world!
```

Congratulations! You have written your first program



Your first program

Type in “print(“Hello world!”)” and press enter. You should see the following

```
>>> print("Hello world!")  
Hello world!
```

Congratulations! You have written your first program

Same result:

```
>>> "Hello World!"  
'Hello World!'
```

Why?

What happens if
we write in the editor?

```
1 # -*- coding: utf-8 -*-  
2  
3 print("Hello world!")
```

```
1 # -*- coding: utf-8 -*-  
2  
3 "Hello World!"
```

Your first program

```
>>> print("Hello world!")  
Hello world!  
>>> print(4+4)  
8
```

Why?

Using Python as a Calculator

- Python has the capability of carrying out calculations
- Standard operators, +, -, * and / work as you would expect; parenthesis () can be used for grouping

```
>>> 3 + 4
7
>>> -2 - 1 + 5
2
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5
1.6
>>> 6 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Using Python as a Calculator

- Other useful operators:
 - `//` : floor division
 - `%` : remainder
 - `**` : powers
- Assign a value to a variable using `"="`
- Last printed expression assigned to the variable `"_"` (use as read-only variable)

```
>>> 17 // 4
4
>>> 17 % 4
1
>>> 5 ** 3
125
```

```
>>> width = 5
>>> height = 3
>>> width * height
15
>>> width + _
20
>>> depth
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'depth' is not defined
```

Quizz time



0. What would be the equivalent code and output if we would use the Editor instead?

a)

```
>>> 17 // 4
4
>>> 17 % 4
1
>>> 5 ** 3
125
```

b)

```
>>> width = 5
>>> height = 3
>>> width * height
15
>>> width + _
20
```

```
1 # -*- coding: utf-8 -*-
2
3 print(17 // 4)
4 print(17 % 4)
5 print(5 ** 3)
```

```
1 # -*- coding: utf-8 -*-
2
3 width = 5
4 height = 3
5 area = width * height
6 print(area)
7 print(area + width)
```

Using Python as a Calculator

- If you want to do more sophisticated calculations then *import* the *math* module. Do not worry about what it means right now, we will cover this later during the course.
- For example, take the square root of a number:

```
>>> 36**.5  
6.0  
>>> import math  
>>> math.sqrt(36)  
6.0
```


Built-in numeric data types

- int: Integer numbers (e.g. 4, 6, -4)
- float: Decimal numbers (e.g. 5.0, -3.1)
- complex: Complex numbers; use j or J to indicate the imaginary part (e.g. 3+5j)

```
>>> (7+5j) + (8-3j)
(15+2j)
```

Operators with mixed type operands convert the integer operand to floating point

Quizz time



1. Fill in the blank to output “eggs eggs eggs”.

```
>>> _____ (“eggs eggs eggs”_____
```

```
>>> print(“eggs eggs eggs”)
```

2. Which option is the output of this code?

```
>>> (4 + 8) / 2
```

a) 6.0 **a) 6.0**

b) 6

c) 8

Quizz time



3. Fill in the blank such that a ZeroDivisionError is caused.

```
>>> (13 + 89) / (-2 + __)
```

```
>>> (13 + 89) / (-2 + 2)
```

4. What is the result of this code?

```
>>> 2 + 3 + 4 + 5.0 + 6  
20.0
```

5b. What is result of this code?

5a. What is result of this code?

```
>>> 7 % (13 // 5) 1
```

```
>>> print(10 + 0.1 - 10)
```

For more details visit

<https://docs.python.org/3.5/tutorial/floatingpoint.html>

Strings

- Use a string if you want to use text in Python
- Strings can start/end with ' or "
- A string is generally displayed using single quotes '

```
>>> "Python is amazing!"  
'Python is amazing!'  
>>> 'Lean back and enjoy this course'  
'Lean back and enjoy this course'
```

Strings

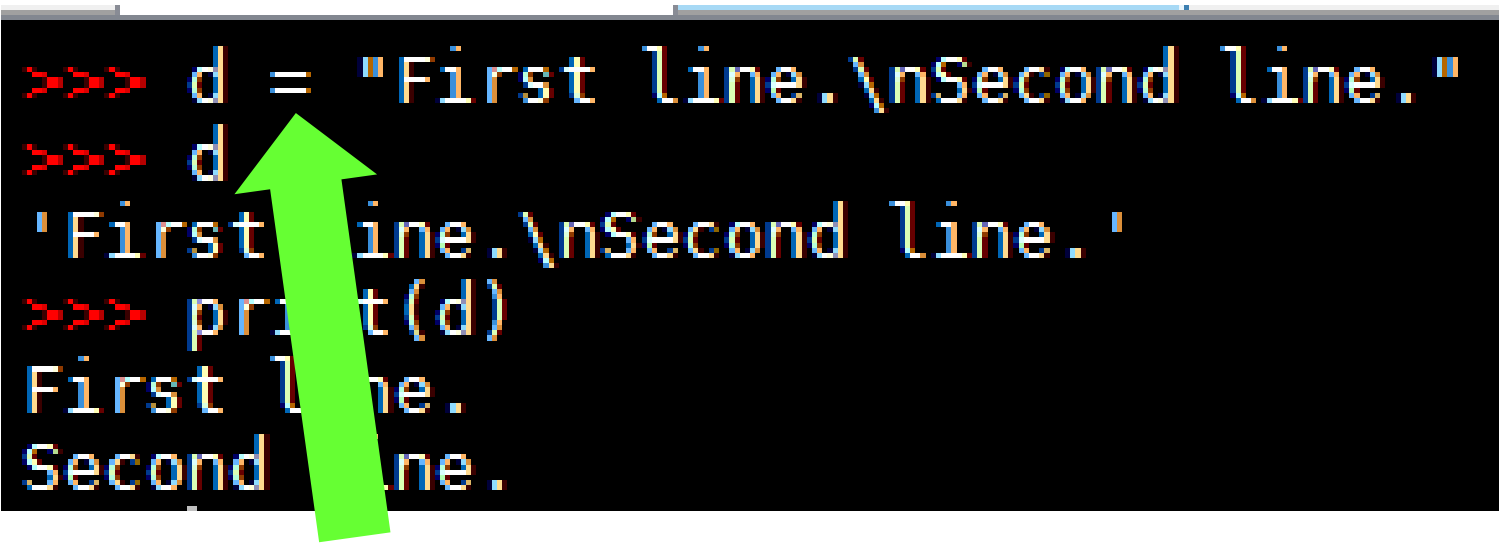
What do you think will happen if we include “ in a string?

- Some characters cannot be directly included in a string
- Escape such special characters using a backslash before them

Escape	What it does.
\\	Backslash (\)
\'	Single-quote (')
\"	Double-quote (")
\a	ASCII bell (BEL)
\b	ASCII backspace (BS)
\f	ASCII formfeed (FF)
\n	ASCII linefeed (LF)
\N{name}	Character named name in the Unicode database (Unicode only)
\r	Carriage Return (CR)
\t	Horizontal Tab (TAB)
\uxxxx	Character with 16-bit hex value xxxx (u" string only)
\Uxxxxxxxx	Character with 32-bit hex value xxxxxxxx (u" string only)
\v	ASCII vertical tab (VT)
\ooo	Character with octal value ooo
\xhh	Character with hex value hh

Output: The print() function

The **print()** function produces a more readable output, by omitting the enclosing quotes and by printing escaped and special characters



```
>>> d = "First line.\nSecond line."  
>>> d  
'First line.\nSecond line.'  
>>> print(d)  
First line.  
Second line.
```

“d” is a **variable** and allows you to store a value using *one equals sign*. More on variables later on.

Input: The input() function

The **input()** function prompts the user for input, and returns what they enter as a string (with contents automatically escaped).

```
>>> input("How are you? ")
How are you? this is what the user enters
'this is what the user enters'
>>> d = input("How old are you? ")
How old are you? 55
>>> d
'55'
```

- The input and print functions are very useful in programs but not at the Python console as input and output are done automatically.

String operations: **Concatenation**

Concatenation is process of adding up strings in Python.

- It does not matter if strings were created with “ or ‘
- Strings containing numbers are added up as strings and not integers
- Error when adding up string to a number (can be avoided with type conversion)

```
>>> "Spam" + 'eggs'
'Spameggs'
>>> print("First string" + ", " + "second string")
First string, second string
>>> "2" + "3"
'23'
>>> 2 + "3"
```

String operations: Concatenation

- However, strings can be multiplied by integers (but not by other strings or floats)

```
>>> print("Hi!" * 3)
Hi!Hi!Hi!
>>> 2 * "3"
'33'
>>> "4" * "9"
```

```
>>> "Hi" * 3.0
```

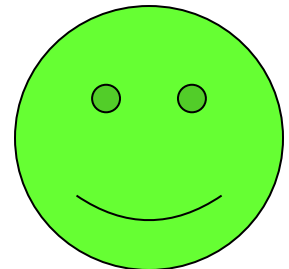
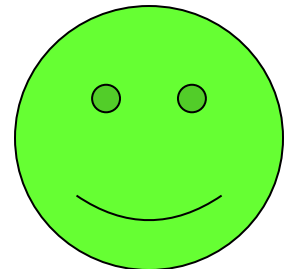
String operations: Concatenation

- Within the `print()` function, concatenation can be achieved via the “+” operator or via “,”
- “,” more convenient as it considers items to be concatenated independently from each other

```
>>> print("Hello "+"world!")
```

OR

```
>>> print("Hello ", "world!")
```

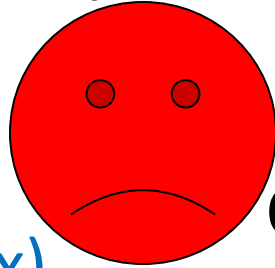


Type conversion

- Another example:

```
>>> x = 5
```

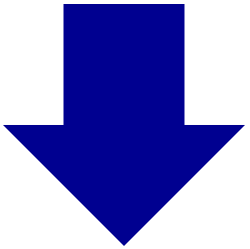
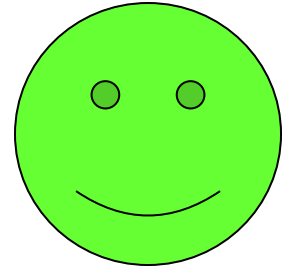
```
>>> print("x = "+x)
```



OR

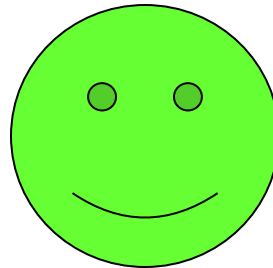
```
>>> x = 5
```

```
>>> print("x = ", x)
```



```
>>> x = 5
```

```
>>> print("x = "+str(x))
```



Quizz time



7. What is the output of this code?

```
>>> print(4 * "2")
```

'2222'

8. What is the correct output of this code?

```
>>> x = 5
```

```
>>> print("x" + " y z")
```

a) x y z

a) x y z

b) 5 y z

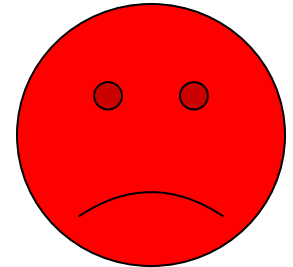
c) Error: can't add string to integer

Quizz time

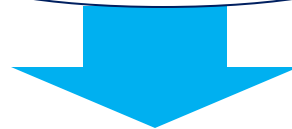


9. Write a small program (using the editor) that asks the user for her **year of birth**, and then output “You will be x years old in 2020”, where x is the actual age of the user in 2020.

→ *Can you do this task with your current knowledge of Python? If not, where are you stuck?*



```
1 # -*- coding: utf-8 -*-  
2  
3 YoB = input("Your Year of birth?")  
4 print("You will be", 2020 - YoB, "years old in 2020")
```



`TypeError: unsupported operand type(s) for -: 'int' and 'str'`

Type conversion

- Some operations are impossible to complete due to type incompatibility

```
>>> "4" * "9"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'str'
>>> "4" + "9"
'49'
```

- Solution to this is **type conversion** (int(), float(), str())

```
>>> int("4") * int("9")
36
>>> int("4") + int("9")
13
```

Quizz time



10. What is the output of this code?

a) `>>> print("4" / "2")`

`TypeError: unsupported operand type(s) for /: 'str' and 'str'`

b) `>>> print("4" + "2")`

`'42'`

11. What is the output of this code?

`>>> float("111" * int(input("Enter a number:")))`

Enter a number: **2**

111111.0

Variables

A **variable** allows you to store a value by assigning it to a name (using *one equals sign*), which can be used to refer to the value later in the program.

```
>>> width = 5
>>> height = 3
>>> width * height
15
>>> width + _
20
>>> depth
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'depth' is not defined
```

Variables

- Variables can be reassigned arbitrarily often to change their value
- **Variables do not have specific types! → dynamic semantics**
- Variable names can consist only of letters, numbers, and underscores (_), and they can't start with numbers
- Python is case sensitive

```
>>> x = 34.5
>>> print(x)
34.5
>>> x = "help"
>>> print(x)
help
>>> print(x * 2)
helphelp
>>> print(x + "!")
help!
```

```
>>> hello_variable = 89
>>> 123_hello_variable = 89
      File "<stdin>", line 1
        123_hello_variable = 89
                                ^
SyntaxError: invalid syntax
>>> hello variable = 89
      File "<stdin>", line 1
        hello variable = 89
                        ^
SyntaxError: invalid syntax
>>> HELLO_variable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'HELLO_variable' is not defined
```

Variables

- Trying to reference a variable you have not assigned to causes an error
- Use del statement to remove a variable (i.e. reference from the name to the value is deleted); deleted variables can be reassigned later as usual

```
>>> hello = "test 1"
>>> hello
'test 1'
>>> world
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'world' is not defined
>>> del hello
>>> hello
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'hello' is not defined
```

- del is rarely used.
- The case where you might want to use del would be if you want to make it *explicitly clear* that this variable cannot be used anymore by this part of the program.

Quizz time



12. What is the output of this code?

```
>>> hello = 3
```

```
>>> world = 2
```

```
>>> del hello
```

```
>>> world = 5
```

```
>>> hello = 4
```

```
>>> print(hello * world)
```


20

In-place operators

In-place operators allow you to write arithmetic operations, such as “`x = x + 4`”, more concisely as “`x += 4`”

- Whatever is on the right hand side of the assignment “=”, gets computed first. Only after that, the result is assigned to the variable on the left hand side.
- In-line operations, e.g. `x += 1` equivalent to `x = x + 1`. Python cannot do increment (`++`) or decrement (`--`) operators

```
>>> x = 5
>>> print(x)
5
>>> x += 3
>>> print(x)
8
```



Identical result

```
>>> x = x + 3
```

In-place operators

In-place operators allow you to write arithmetic operations, such as “`x = x + 4`”, more concisely as “`x += 4`”

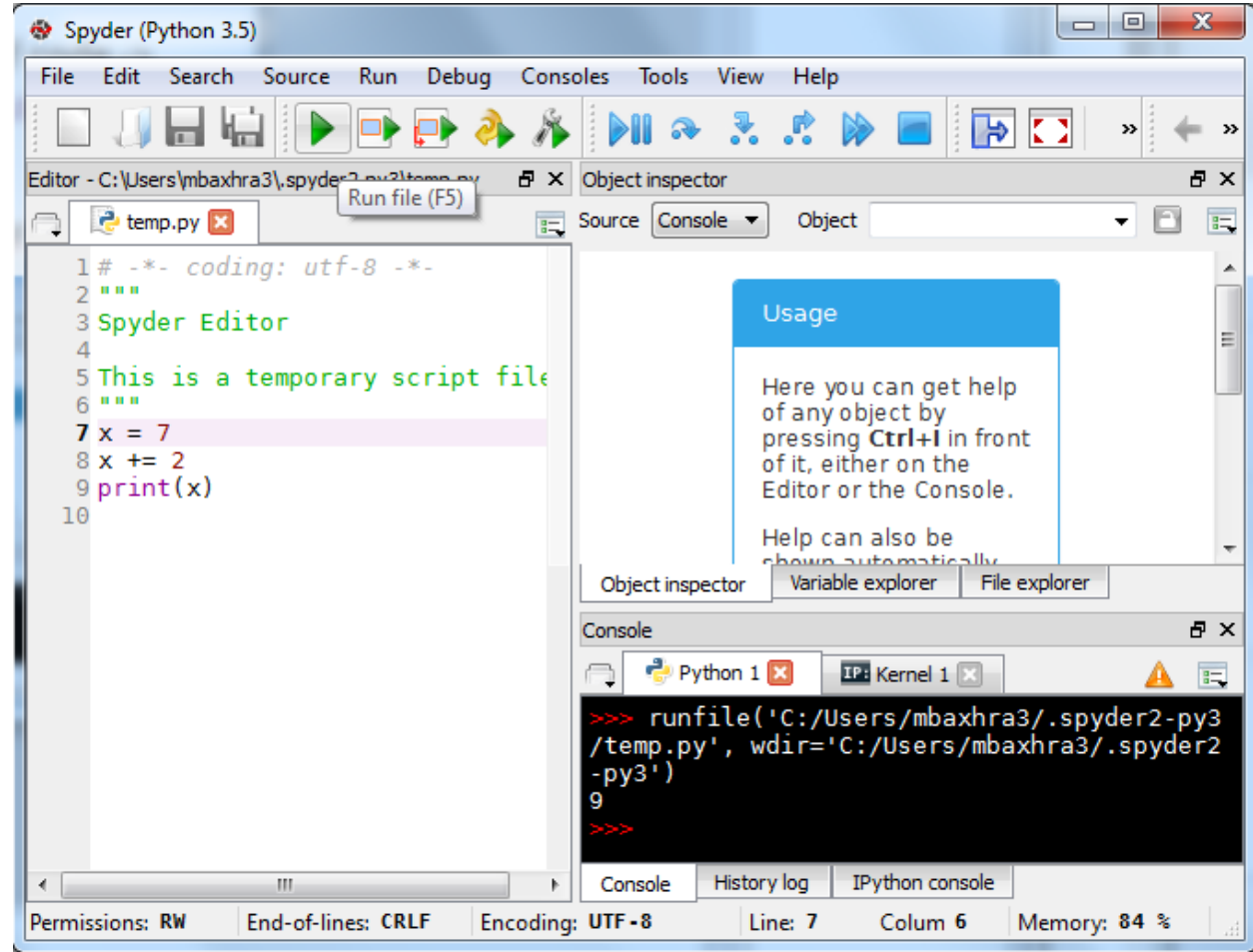
- Also possible with other operators, such as `-`, `/`, `*`, `%`.
- Operators can also be applied to types other than numbers, e.g. strings

```
>>> x = 5
>>> print(x)
5
>>> x += 3
>>> print(x)
8
>>> x /= 4
>>> print(x)
2.0
```

```
>>> var = "hello"
>>> print(var)
hello
>>> var += " world"
>>> print(var)
hello world
```

Using an editor

- Normally, programs consist of many lines of code -> write lines of code in a file and execute all using a Python interpreter
- Python source files have an extension of **.py**



Recap

- Integrated Analysis Environment
- Python essentials
 - Your first program → `print()`
 - Using Python as a Calculator → `5**2`, `5%3`, `math.sqrt(5)`, etc
 - Numeric data types → `int`, `float`, `complex`
 - String operations → `input()`, concatenation
 - Type conversion → `int("4")`, `float("8")`
 - Variables → `var1 = 4`, `var1 += 3`

Week 1/ Lecture 2 Preparation

- Prior to seminars
 - Revise lecture slides and do quizzes again
 - Have a go at quizzes and questions on BB
- Prior to lecture, have a look at
 - built-in types (sequences, mappings)
 - control flow statements (for, if, while loops)
 - comparison and Boolean operations
- BB: Additional materials
 - Multiple choice questions
 - Videos to support you in understanding how to determine and interpret line equations