MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Computer Science Department

COMP8060 – Scientific Programming with Python

Week 2 – Intro to Python

Dr. Bruno Andrade

September, 2025

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

- **Summary:**
  - Python and its runtime environment
  - Input/Output and Processing in Python
  - Data Types
  - Operators

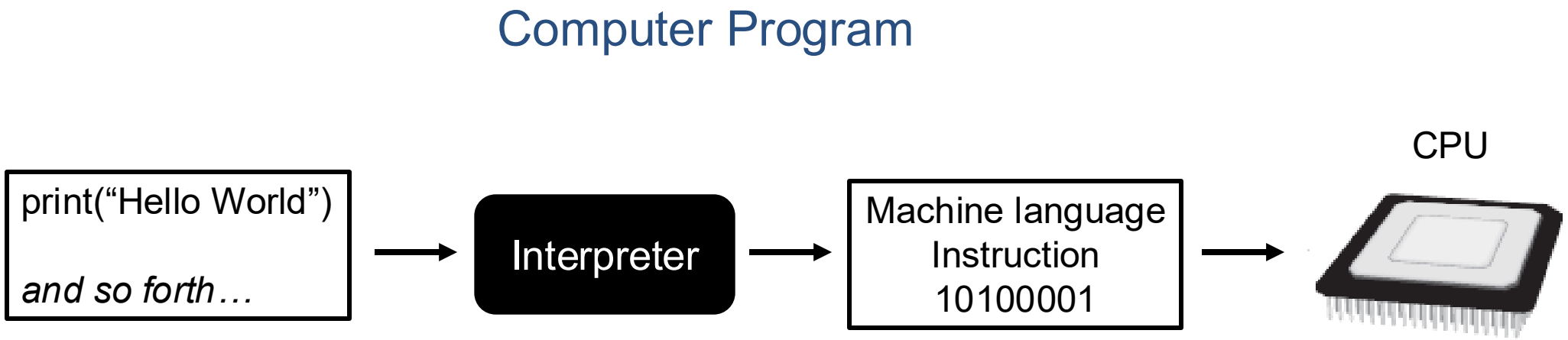
MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Python and its runtime environment

Python

- Was first released in 1991.
- Multi-paradigm programming language.
- Weapon of choice of Data Scientists.
- Constantly updated.
- High level (Interpreted) language.

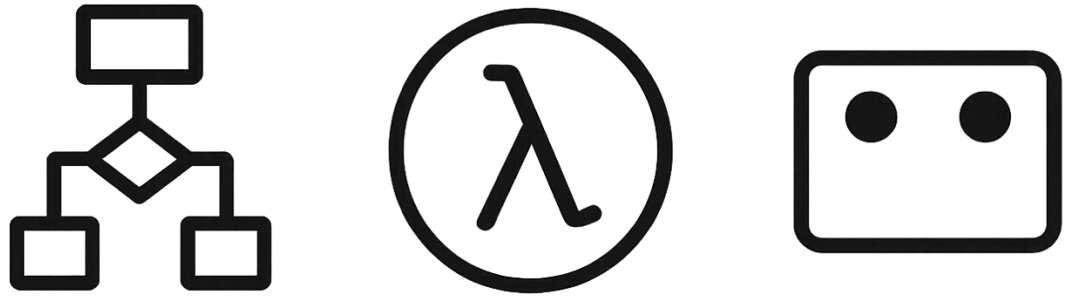
Python is an Interpreted Language



Python is an Object-oriented language

- Which means that everything in python are objects, containing data and what to do with that data. This is one of the so called “Programming paradigms”.

- There are several of them, but we will learn four of them throughout the semester:



Structured Functional Object-Oriented

Using Python

- You can download a version of Python at http://www.python.org/download/
  - Current stable version is Python 3.12
  - You can also install it using Conda.

- We can make use of a range of integrated development environment (IDEs) to code.
  - IDLE (which is bundled with Python itself)
  - Spyder
  - PyCharm
  - Visual Studio Code
  - Etc…

Jupyter

Jupyter

- Jupyter is an open-source application that allows you to create and share documents that contain live code.
- Go to http://jupyter.org/install and follow the instructions.

JupyterLab

Install JupyterLab with pip:

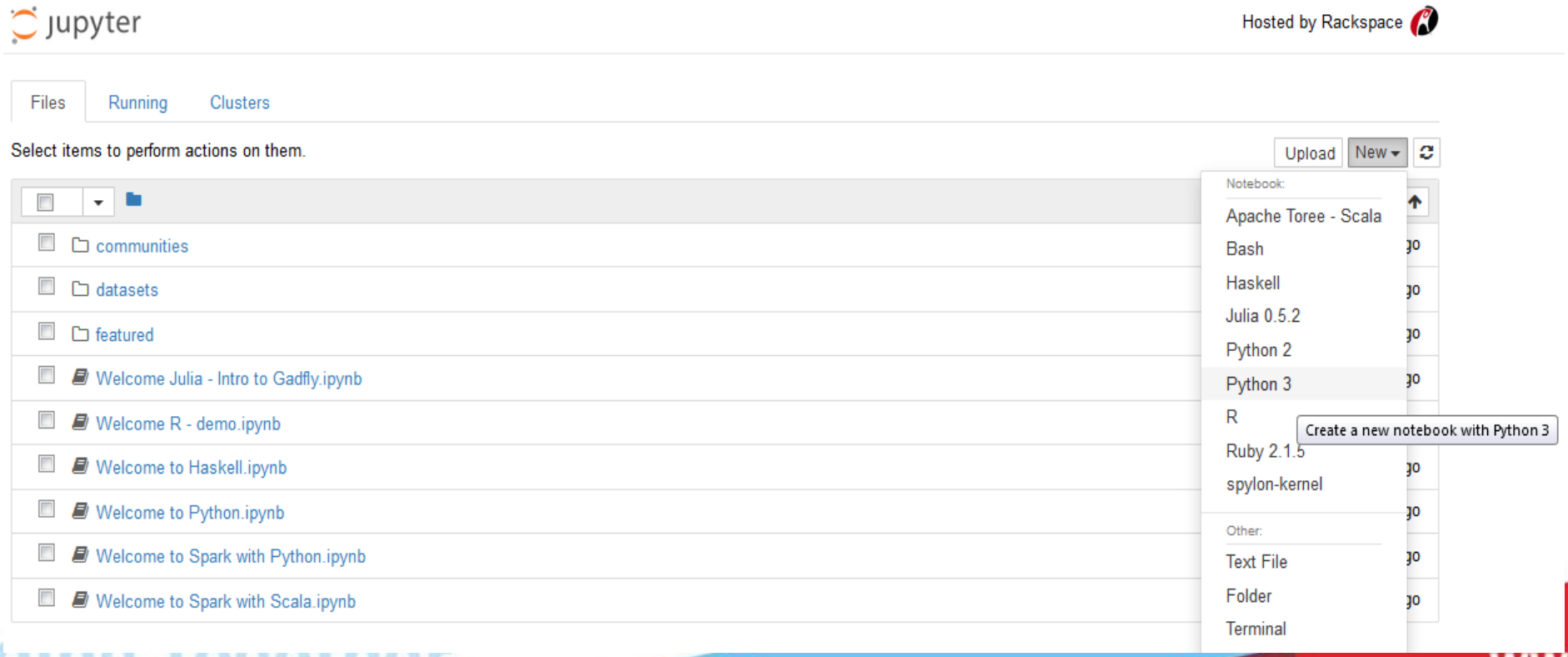
```
pip install jupyterlab
```

Install JupyterLab with conda:

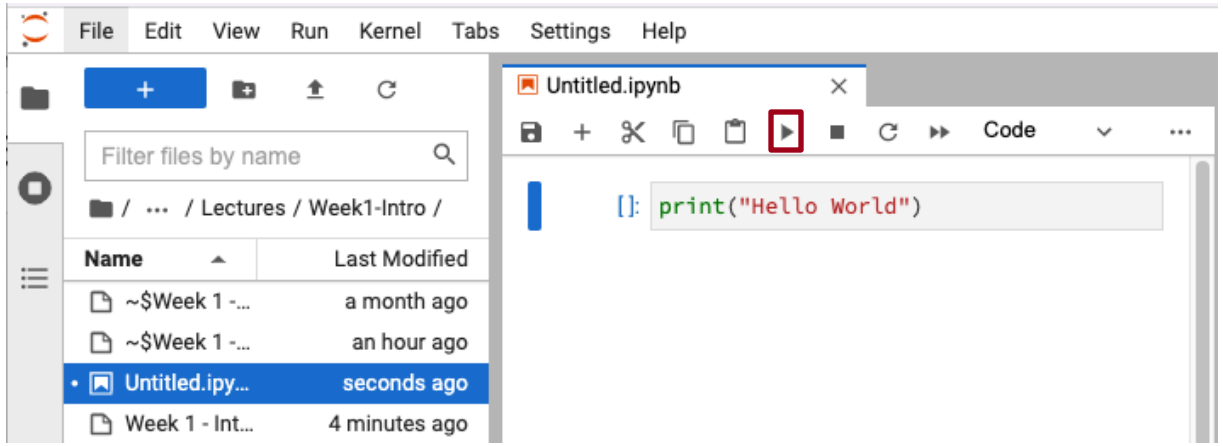
```
conda install -c conda-forge jupyterlab
```

Jupyter Notebooks

- You can create a Jupyter notebook for a range of programming languages.
- It's a great way for learning a language as it provides you with an interactive shell that allows you to type/run commands and see the output.



Jupyter Notebooks



- The screenshot above shows my notebook and I've typed a simple command print ("Hello World").

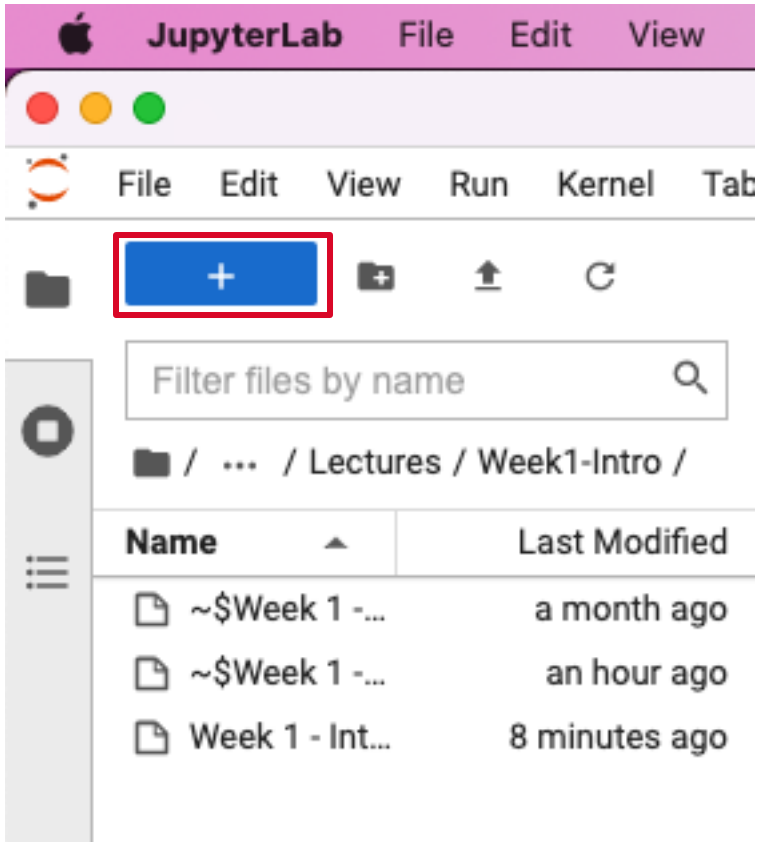
```
[1]: print("Hello World")

     Hello World
```

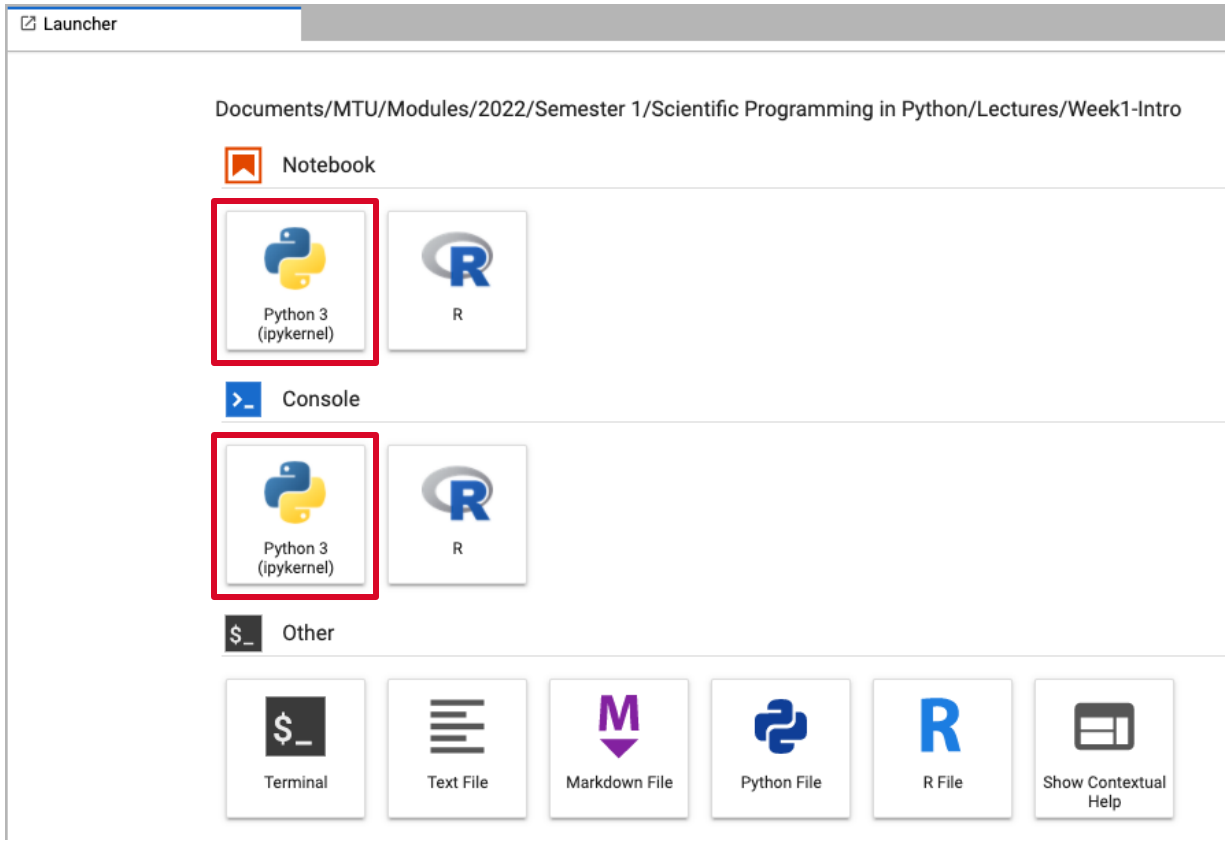
Jupyter is not suitable for serious software development.

Jupyter Notebooks – Google colab

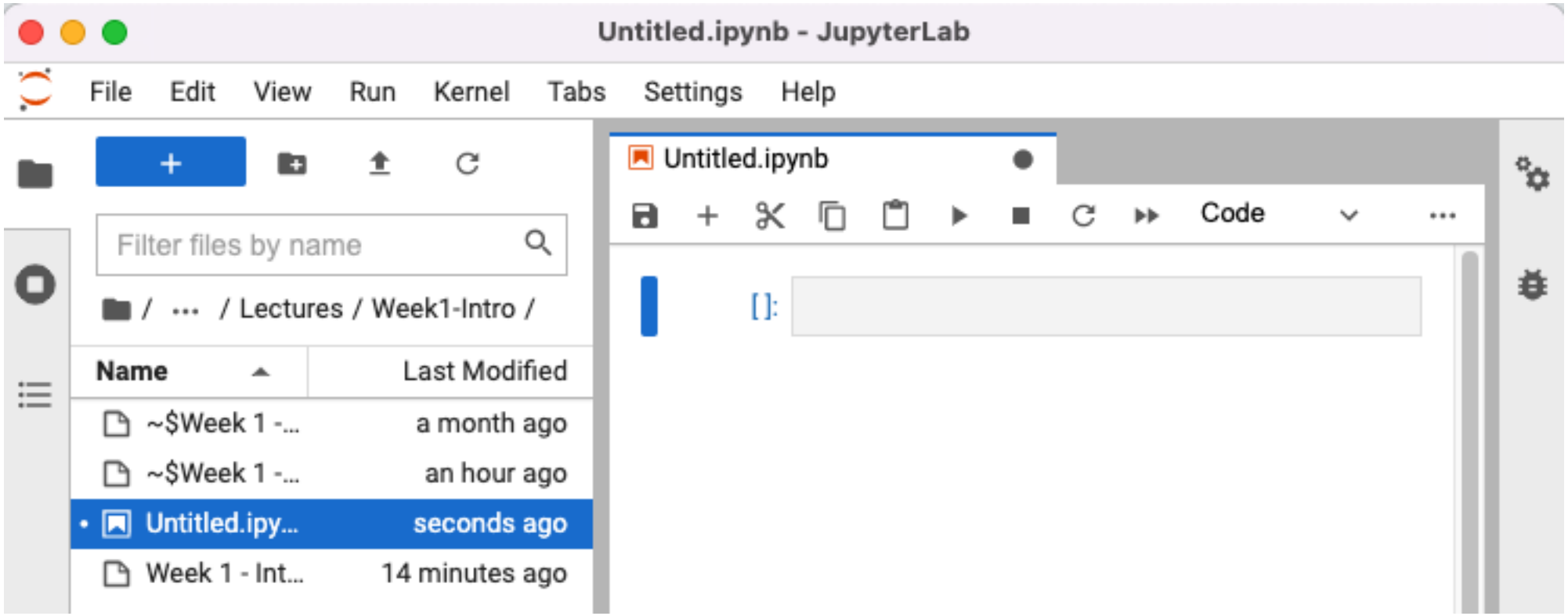
- Why am I insisting in Jupyter, although it's not the best tool for development?
- It’s the best tool for collaboration, you guys can code together using google colab!!!!

Jupyter Notebooks – Creating a file



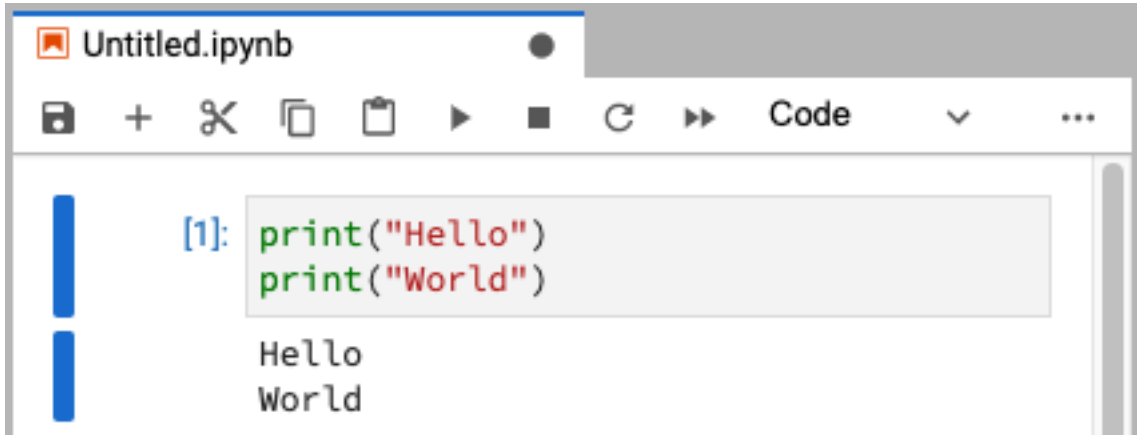


Jupyter Notebooks – Creating a file



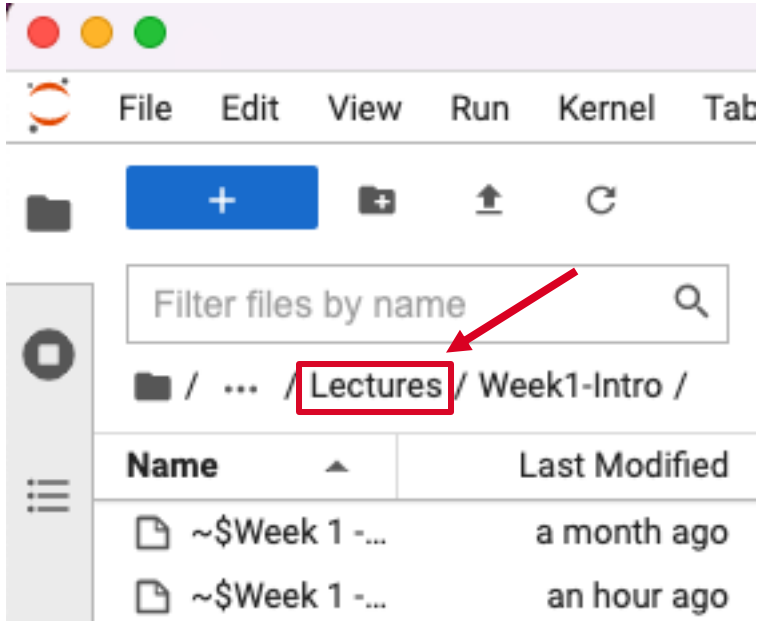
Multiple Lines of Code and Sequence of Execution

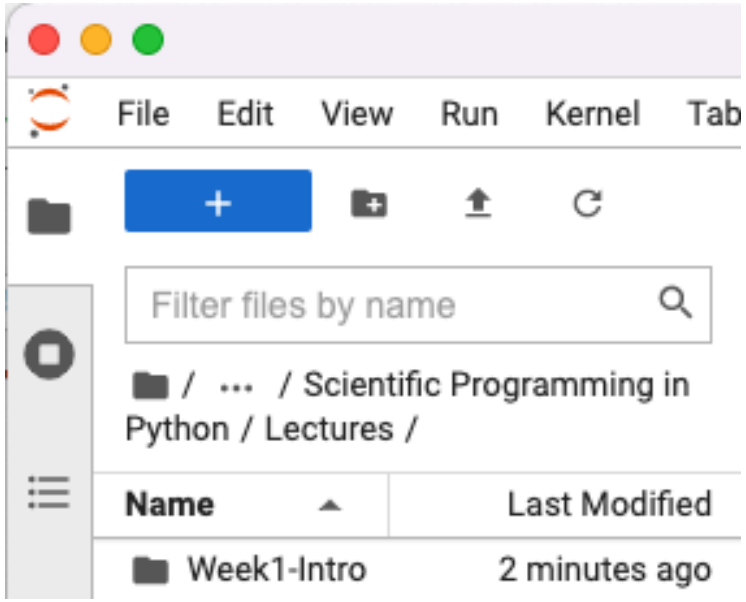
- We can place multiple lines of code into our program.
- When we run the program the interpreter starts at the top of the file and executes statements from top to bottom.



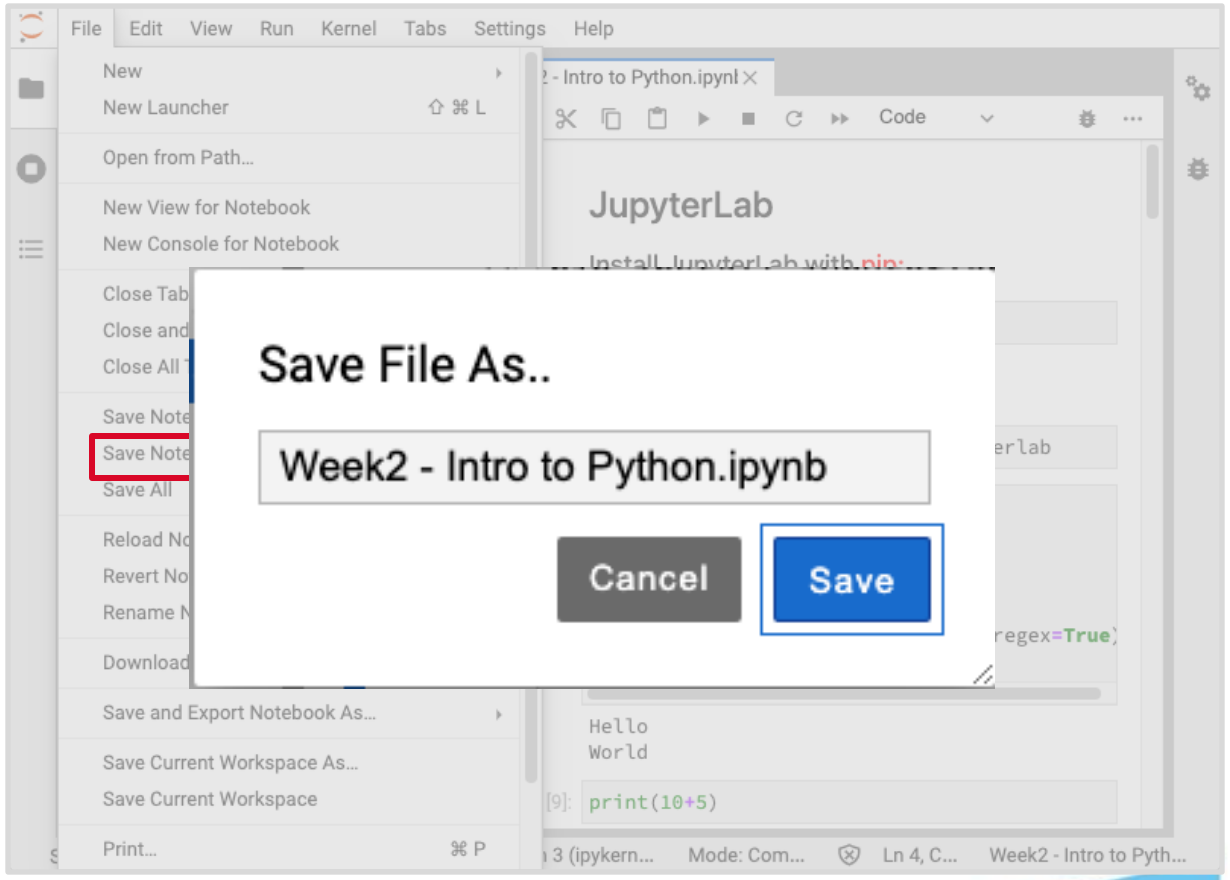
Working Directory

- You can easily change the current working directory.





Saving notebook



Useful Shortcuts

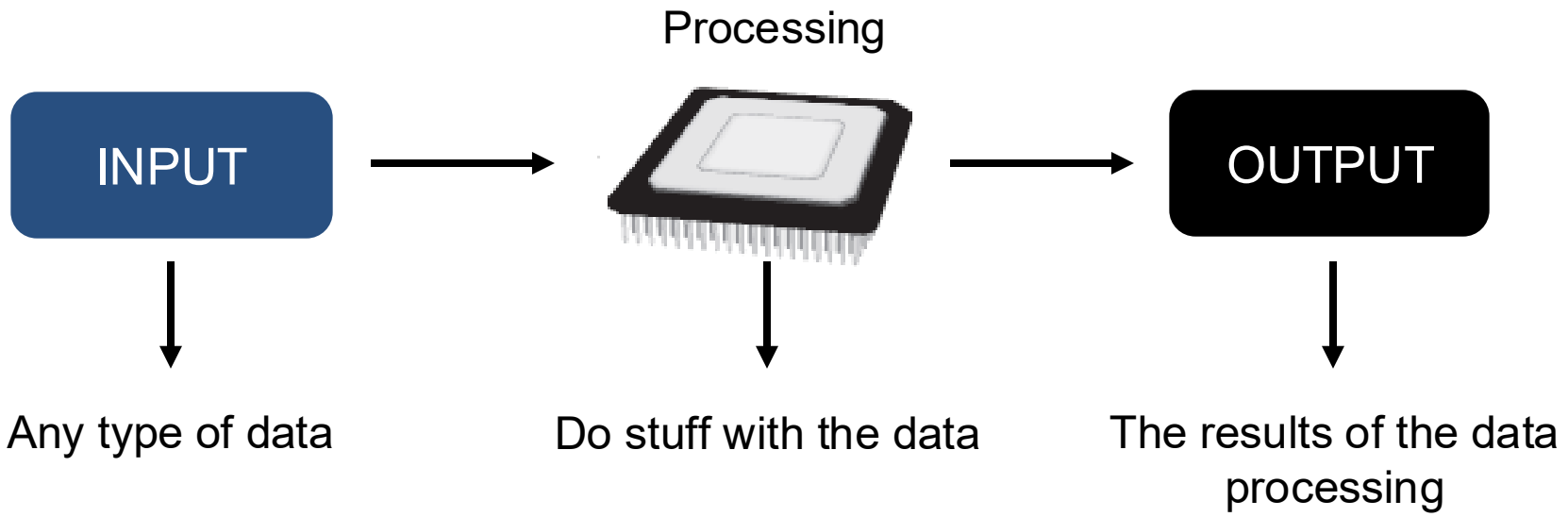
- CTRL+ENTER executes the current cell.
- ALT+ENTER executes the current cell and creates a new cell.
- Auto-complete is your best friend, use Tab to complete functions and variables names.

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Input/Output and Processing in Python

Input, Processing, and Output

- Typically, computer programs perform a three-step process



Displaying Output with the `print` Function

Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

- The function **print**(“Hello World”) is simply an instruction to the interpreter to print the **String** “Hello World”.

- String: sequence of characters that are used as data (example ‘Hello World’)
  - Must be enclosed in single (‘) or double (“) quotation marks
  - ‘hello world’ is equivalent to “hello world”

```
print("Hello World")
```

```
Hello World
```

What just happened?

```
print("Hello world")
```

Python function

Python interpreter

Hello World

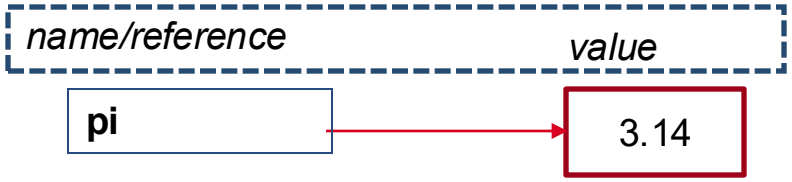
A function is a reusable code.

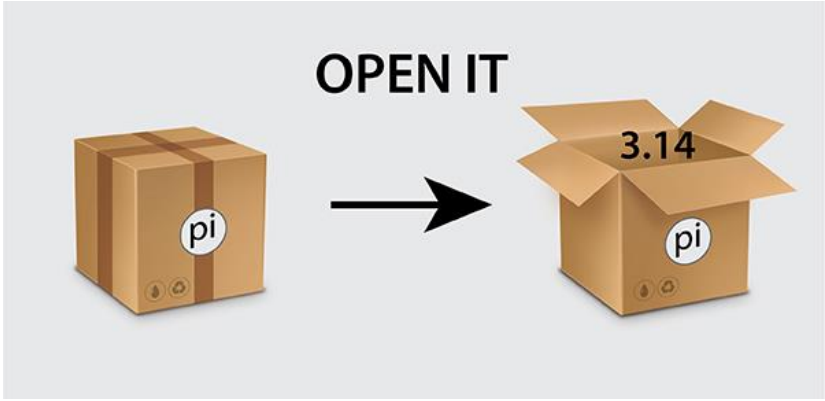
Methods are functions tied to an object.

```
def my_print(*args, sep=' ', end='\n', file=None, flush=False):
    """
    Simplified version of Python's print function
    """
    # Default to sys.stdout if no file is specified
    if file is None:
        import sys
        file = sys.stdout

    # Convert all arguments to strings and join with separator
    output = sep.join(str(arg) for arg in args)

    # Add the ending
    output += end

    # Write to the specified file
    file.write(output)

    # Flush if requested
    if flush:
        file.flush()


# Example usage
my_print("Hello", "world!", sep=", ", end="!\n")
# Output: Hello, world!!
```

Variables

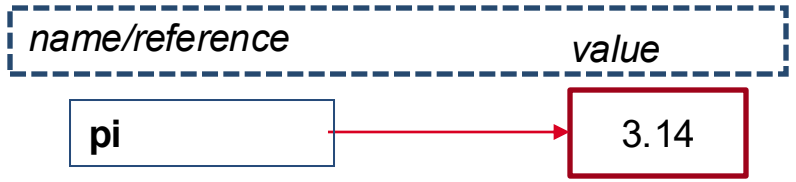
- **Variable**: name that represents the data stored in the computer memory.
  - Used to access and manipulate data stored in memory
  - Think of it as a box. The 'box' needs a name/label. You can refer to it later by this name and print it out.





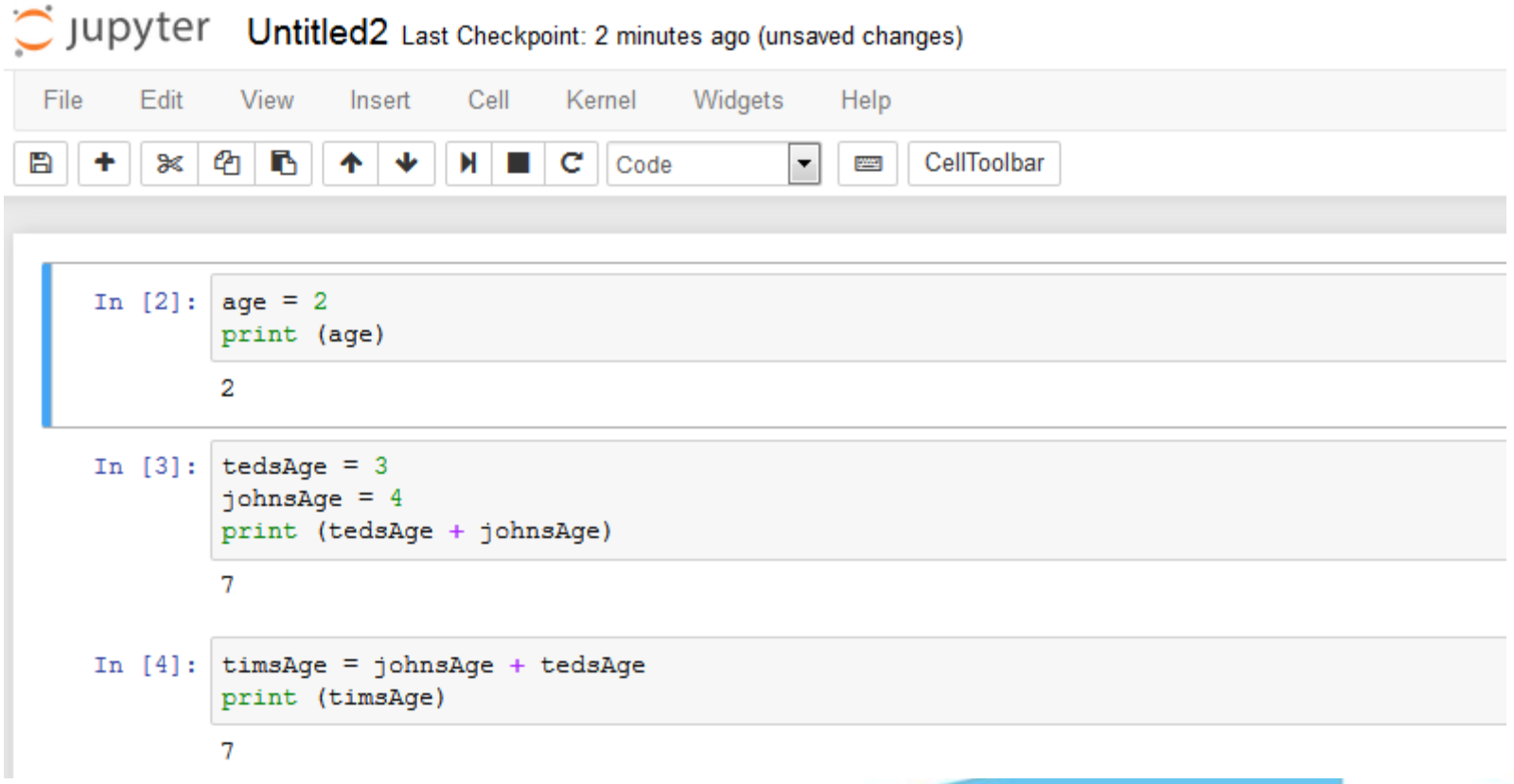
Variables

- Assignment of a value to a variable: used to create a variable and make it reference data
  - General format is ***variable = information***
  - Example: pi = 3.14 → Assignment operator!
  - Variable names are case-sensitive.



Variables

- A variable references the value it represents



Displaying Multiple Items with the `print` Function

- Python allows you to display multiple items with a single call to `print`
  - Items are separated by commas when passed as arguments
  - Arguments displayed in the order they are passed to the function

```
In [7]: age = 105
        name = "ted"
        print ("name is ", name, " age is ", age)

        name is  ted  age is  105
```

Data Output

- Instead of using this type of output notation, we will use the “f notation”
  - Was introduced in Python > 3.6 and it makes your life easier!
  - The f notation allows you to include variable values inside an output string without breaking a sweat to maintain the template!

```
age = 105
name = "Ted"

print("Name is ",name, "and age is ", age)
```

```
Name is  Ted and age is  105
```

```
age = 105
name = "Ted"

print(f"Name is {name} and age is {age}")
```

```
Name is Ted and age is 105
```

Variable Naming Rules

- Rules for naming variables in Python:
  - Variable name cannot be a **Python keyword**
  - Variable name cannot contain **spaces**
  - First character must be a **letter** (or an underscore)
  - Variables names are **case sensitive**

Variable Naming Rules

- camelCase naming convention
  - The variable name begins with lowercase letters.
  - The first character of the second and subsequent words is written in upperCase.
  - An upper camelCase should be used to name Classes (OOP)

- Underscore (snake_case) naming convention
  - The variable name begins with lowercase letters.
  - _ separates each word in variable_name.

- Variable name should reflect its use
  - bankBalance as opposed to b or bba

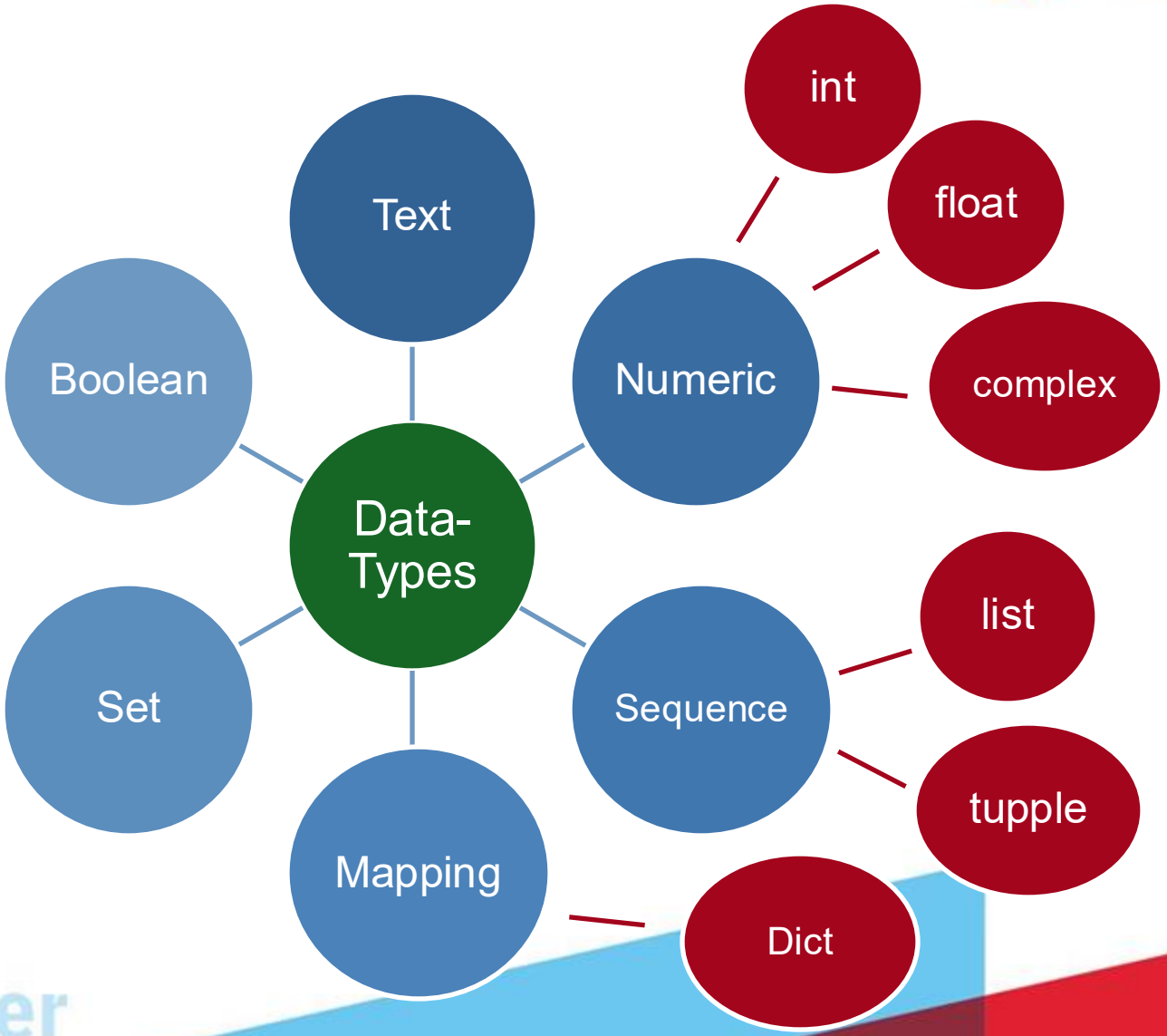
We will follow the PEP8 convention, check the details in here – PEP8

Stick to the f-notation and naming rule, not following the best practices will result in points deductions.

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Data Types

Data Types


Data-Types
Text
Boolean
Numeric
int
float
complex
Set
Sequence
list
tupple
Mapping
Dict

Data Types

- In your program you may need to use different types of data such as integer numbers, strings, booleans, etc.
  - Some of the code you will use may only accept variables of a particular type.

- Python reserves space in memory to store a particular value. However, based on the type of the variable the space allocated can vary.

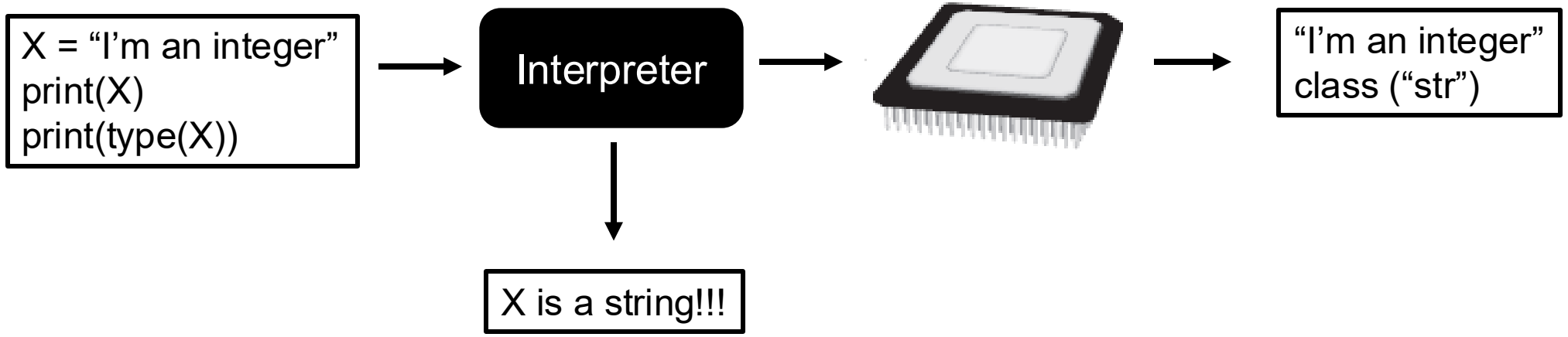
  Variable = 555 – 28 bytes

  Variable = “555” – 52 bytes

- So how does Python decide the type of a variable?
  - Python is not a strong typed language

Data Types

- Python is a weak typed language.
- The interpreter determines the data type of the variable based on its value.



Data Types

- If I create a numeric variable it determines its data type according to the following rules:
  - A numeric literal that is written as a whole number with no decimal point is considered an int. Examples are 7, 12 4, and - 9.
  - A numeric literal that is written with a decimal point is considered a float. Examples are 1.5, 3.14159, and 5.0.

Data Types (int and float)

- So, the following statement causes the number 503 to be stored in memory as an int
  - ***room = 503***
- The following statement causes the number 2.75 to be stored in memory as a float:
  - ***dollars = 2.75***
- The following will store the Strings John and Tim as a Strings:
  - ***firstName = 'John'***
  - ***secondName = "Tim"***

Mixed-Type Expressions

- Data types resulting from math operations depends on the data types of operands
  - Two `int` values: result is an `int`
  - Two `float` values: result is a `float`
  - But what would happen with an operation between an `int` and `float`?
    - The result will be a float.

When you store an item in memory, it is important for you to be aware of the item's data type.

Data Types

- Once you create a variable, that variable is not permanently tied to a specific data type.
- The same Python variable can be assigned to different variables

```
# A variable is not permanently tied to a specific data type.

timsAge = "five"
print (timsAge)

timsAge = 5
print (timsAge)

timsAge = 5.5
print (timsAge)
```

```
five
5
5.5
```

Boolean Data Type

- Python also provides a Boolean datatype, that is a variable that is either True or False (True and False are both **keywords** in Python)

```
pi = 3.14
print(pi)
```

```
3.14
```

```
pi==3.14
```

```
True
```

```
type(pi==3.14)
```

```
bool
```

Checking a Variable Datatype

- If you are unsure of the data type of a variable you can use a in-built function called *type()*.

```
number = 15
name = "scully"

print (type(number))
print (type(name))
```

```
<class 'int'>
<class 'str'>
```

Every datatype is an object!

- And as such they have methods associated to it.
- And there are other ways to discover them without referring to the documentation. Curious? Just use tab!!

```
phrase = "I have a dream"
phrase.
```

f	capitalize	function
f	casefold	function
f	center	function
f	count	function
f	encode	function
f	endswith	function
f	expandtabs	function
f	find	function
f	format	function
f	format_map	function

```
phrase = "I have a dream"
phrase.casefold()
```

```
'i have a dream'
```

Reading User Input from the Keyboard

- Most programs need to read input from the user
- Python's input function is useful for reading input from the keyboard.
- It reads input as a String
- Format: ***variable* = input(*prompt*)**

```
name = input("What is your name?")
print (name)
print ( type(name))
```

```
What is your name?Ted
Ted
<class 'str'>
```

Reading User Input from the Keyboard

- The problem with the input function is that we may not always want to read string variables in from the user.
- Have a look at the following code and see if you can determine the source of the problem.

```
age = input("What is your age")
print (age+2)
```

```
What is your age 13

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-38672b2b662f> in <module>()
      1 age = input("What is your age")
----> 2 print (age+2)

TypeError: must be str, not int
```

Reading User Input from the Keyboard

```
age = input("What is your age")
print (age+2)
```

```
What is your age 13

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-38672b2b662f> in <module>()
      1 age = input("What is your age")
----> 2 print (age+2)

TypeError: must be str, not int
```

- The problem is the code below is that we have read in the user input as a string value.
- The variable age stores the string value “13”.
- We then attempt to add the numerical value 2 to the string value “13”. You cannot add a string and an int.

Reading String input

- Python has a number of methods that allows us to force the conversion from one data type to another. This process is also called casting.
- It provides the following:
  - int(), which converts a value to an int (for example, it can convert a string to an int)
  - float(), converts a value to a float (for example, it can convert a int to a float)
  - str(), converts to a string data type

Reading String input

- If we pass the String variable age to the int function, it returns a new int value for age.

```
# Notice we convert age from being a string variable to an int variable using the int function.
age = input("What is your age")
age = int(age)
print (age+2)
```

```
What is your age 13
15
```

Data Type Conversions

- When a float is converted to an int, any fractional part is thrown away, or truncated.

```
realNumber = 23.45
print(realNumber, "is of type ", type(realNumber))

number = int(realNumber)
print(number, "is of type", type(number))
```

```
23.45 is of type  <class 'float'>
23 is of type <class 'int'>
```

Data Type Conversions

- If the user has entered integer values for both books and months, this statement will perform integer division.

- If you want the result to be completely accurate, you need to make sure that at least one of the operands in the division operation is a float.

- What do you guys think will happen if we convert an int to a float?

Like the number 15?

15.0

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Operators

Operators

- Operators are special symbols that carry out arithmetic or logical operations

Arithmetic

- Arithmetic operators are used with numeric values to perform common mathematical operations:

Symbol	Operation	Description	Example a=3, b=2:
+	Addition	Sum	x = a+b ; x =5
-	Subtraction	Subtract	x = a-b ; x=1
*	Multiplication	Multiply	X = a*b ; x = 6
/	Division	Returns quotient of division	X = a/b ; x =1.5
%	Remainder	Returns Remainder of division.	X=a%b ; x = 1
**	Exponent	Raises a number to a power.	X=a**b ; x = 9

```
x = 5
y = 3
print(x+y)    8
print(x-y)    2
print(x*y)    15
print(x/y)    1.66666
print(x%y)    2
print(x**y)   125
```

Precedence of operators

- Python follows the same rules you learn in algebra:
  - Sum and Subtraction have the lowest and same precedence
  - Multiplication has the highest precedence.
  - Divisionhas a lower precedence than multiplication.
  - Parenthesis will force the precedence of an operator.

Same precedence operators execute from left to right.

Example of Precedence

```
>>> 12.0 + 6.0 / 3.0
```

1. Division
2. Sum

```
>>> 1+2*2%3
```

1. Multiplication
2. Modulus
3. Sum

Example of Precedence - Use of Parenthesis

```
>>> (12.0 + 6.0) / 3.0
```

1. Sum
2. Division

Result is 6.0

```
>>> (1+2)*2%3
```

1. Sum
2. Multiplication
3. Modulus

Result is 0

The remainder is typically used to detect odd and even numbers!

Assignment

- Assignment operators are used to assign values to variables:

Symbol	Operation	Description	Example:	Equivalent to:
=	Assign	Assign a value	x = 5	…
+=	Increase	Assign a self value plus a number	x += 3	x = x +3
-=	Decrease	Assign a self value minus a number	x -= 3	x = x -3
*=	Multiply	Assign a self value times a number	x *= 3	x = x *3
/=	Divide	Assign a self value divided by a number	x /= 3	x = x/3
**=	Exponent	Assign a self value on the power of a number	x **=3	x = x**3

```
x = 5

x += 3

print(x)
```

```
8
```

Comparison

- Comparison operators are used to compare two values, return a Boolean value

Symbol	Operation	Example:	Equivalent to:
==	Equal	x ==5	Is x equal 5?
!=	Not equal	x != 3	Is x different from 3?
>	Greater than	x > 3	Is x greater than 3?
<	Less than	x < 3	Is x less than 3?
>=	Greater or equal to	x >= 3	Is x greater or equal 3?
<=	Less or equal to	x <=3	Is x less or equal 3?

```
x = 5

print(x==2)    False
print(x!=2)    True
print(x>2)     True
print(x<5)     False
print(x>=5)    True
print(x<=5)    True
```

Logical

- Logical operators are used to combine conditional statements:

Symbol	Description	Example:
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

```
x = 5

print(x > 3 and x < 10)
```

```
True
```

```
x = 5

print(x > 3 or x < 4)
```

```
True
```

```
x = 5

print(not(x > 3 and x < 10))
```

```
False
```

Identity

- Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Symbol	Description	Example:
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

```
identity1 = "This is a string"
identity2 = 10

identity1 is identity2
```

```
False
```

```
identity1 is not identity2
```

```
True
```

Membership

- Membership operators are used to test if a sequence is presented in an object:

Symbol	Description	Example:
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

```
x = "I don't want to miss a thing"

"miss" in x
```

```
True
```

```
x = "I don't want to miss a thing"

"miss" not in x
```

```
False
```

Membership is great to test if an element or value is inside another element, be a string, list or even a table.

Comments

- Comments: notes of explanation within a program
  - Ignored by Python interpreter
    - Intended for a person reading the program's code

- There are two ways of commenting your code in Python:
  - In Python a comment begins with a # character. Everything after the # on the same line is ignored by the interpreter
  - Comments spanning more than one line are achieved by inserting a multi-line string (with """ as the delimiter on each end) also known as a docstring.

Comments

```
# This program displays a person's
# Christian name and surname.
print (‘Diarmuid’)
print (“Grimes”)      # This line prints the surname
```

```
"""
The purpose of the code below is
to .....
"""
print ("hello") # prints the string hello
print ("there")
```

Docstrings are important for functions only, I will show you how to effectively use it when we get there.

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Errors in Python

Error Messages in Python

- When you execute a Python program the interpreter takes each line in turn, turns it into machine code, which is then executed.
- If the interpreter encounters an error it will **output an error message**.

```
number = 5
print (Number + 3)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-9a7e20361ef9> in <module>()
      1 number = 5
----> 2 print (Number + 3)

NameError: name 'Number' is not defined
```

Sample Error Messages

- You cannot use a variable until you have assigned a value to it.
- Clearly the problem on the previous slide is that I'm trying to print the variable Number but I haven't assigned it a value
  - Always look for the line number
  - Gives a basic description of error
  - Takes practice in order to identify the source

Sample Error Messages

Some errors can be difficult to identify and this process takes practice.

```
num = 4
print ("The value of num is " num)
```

```
  File "<ipython-input-11-a8b2df75ee99>", line 2
    print ("The value of num is " num)
                                    ^
SyntaxError: invalid syntax
```

```
num = 4
print ("The value of num is ", num)
```

```
The value of num is  4
```

We will learn how to proper handle errors, and not only identify them, in a couple of weeks.

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

That’s all folks!