MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Computer Science Department

COMP8060 – Scientific Programming with Python

Week 3 – Decision Structures

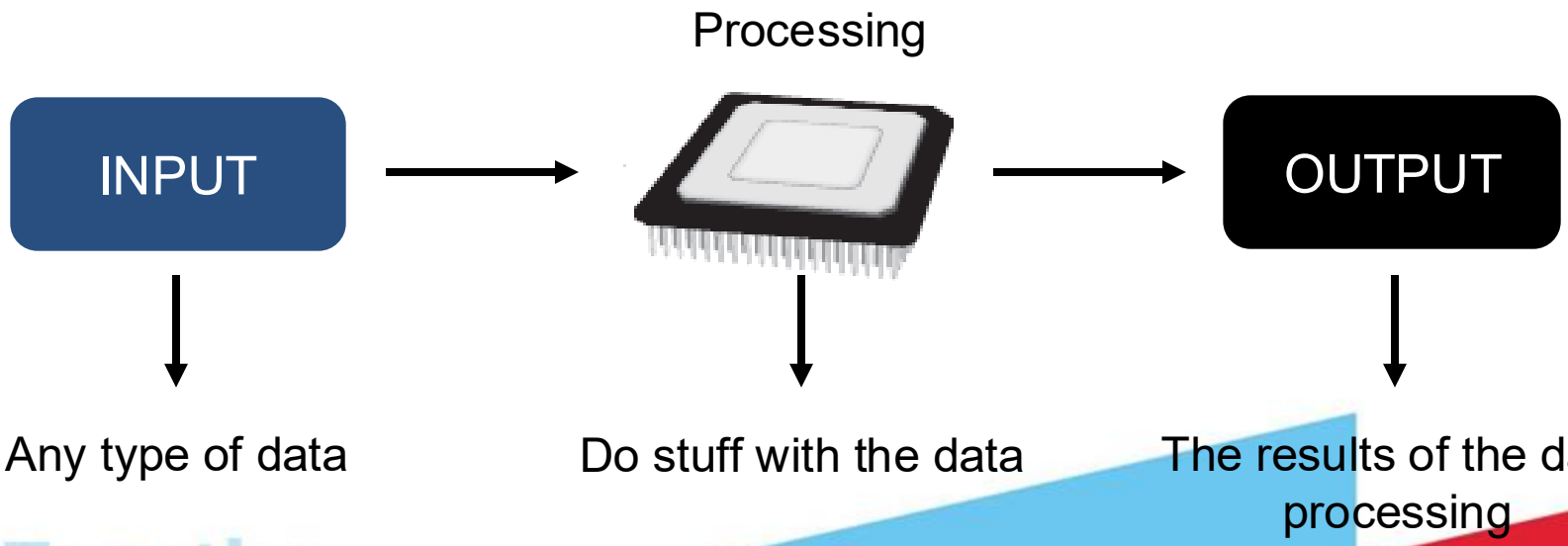
Dr. Bruno Andrade

September, 2025

www.mtu.ie

Last week

- Python and its runtime environment
- Input/Output and Processing in Python
- Data Types
- Operators
- Comments



Assignment (next week).

- The assignment will be delivered next week during the labs.
- It’s going to be a 20 questions long Multiple-choice test, and you will have 1 hour to finish it.
- Each group will do the assignment on its own lab, but don’t worry, all questions are random, so even if your colleague share his answers with you will have 0 effect (which I don’t expect to occur).

- ONLY LINUX!!!

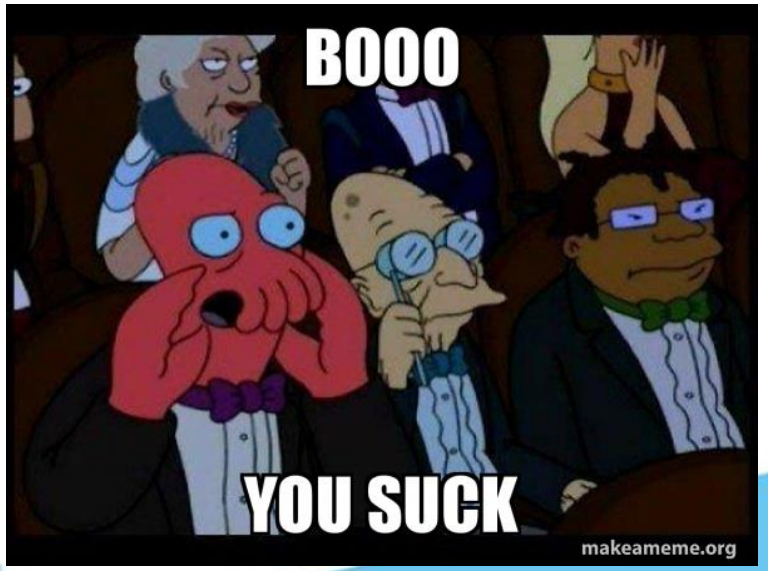
MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

- **Summary:**
  - The if Statement
  - The if-else Statement
  - Nested Decisions Structures and the if-elif-else Statement
  - Logical Operators in Decision Structures
  - Boolean Variables
  - Python Modules
  - Q&A

The if statement

A Decision Structure – let's start with a bad programmer joke.

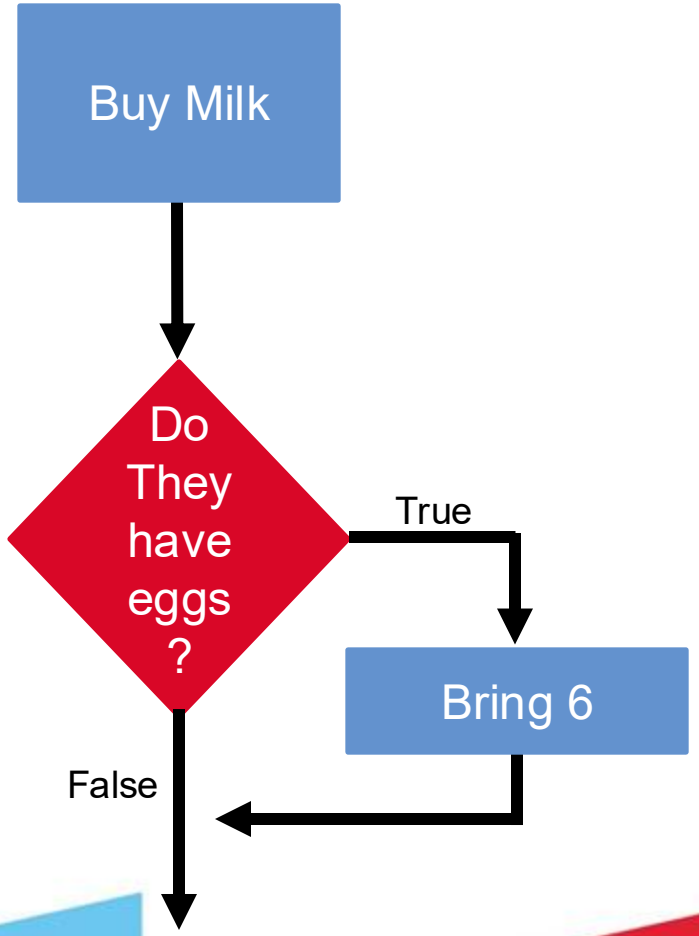
- "Husband, go to the market and buy a bottle of Milk. If they have eggs, bring six.
- The guy, who by the way was a programmer, bought six bottles of Milk.
- The wife baffled asked "Why the hell did you buy six bottles of milk?"
- The husband replied "because they had eggs…"



A Decision Structure

- Decision structure: This bad joke illustrates the basis of a decision structure. In which action(s) are performed only if a condition exists

  - Example:
    - If a user is 18 yr of age or less a ticket will cost them €10.
    - If the room temperature drops below 10 degree Celsius the heating should be activated

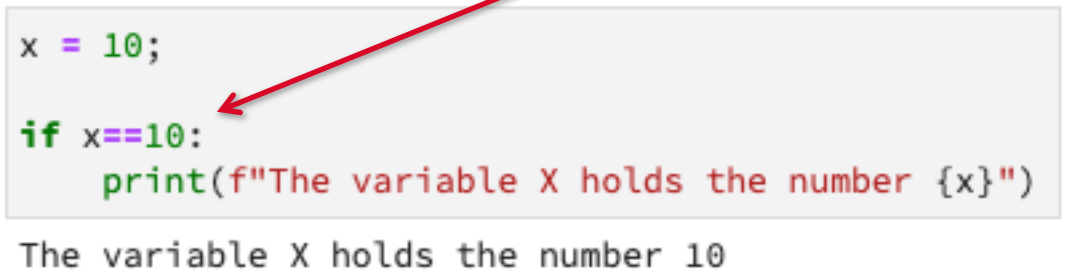


The `if` Statement

: after the condition

The indentation after the if condition is mandatory

- Python syntax:



```
x = 10;

if x==10:
    print(f"The variable X holds the number {x}")
```

```
The variable X holds the number 10
```

- First line is known as the if clause
  - Includes the keyword if followed by a condition
    - The condition can be **true** or **false**
    - When the if statement executes, the condition is tested, and if it is true the **indented block statements are executed**. otherwise, block statements are skipped

What is a Condition (Boolean Expression)

- A condition is a boolean expression that is tested by the *if* statement to determine if it is true or false
  - Example: a > b
    - This expression returns `True` if *a* is greater than *b*; `False` otherwise

Expression	Meaning
x > y	Is x greater than y?
x < y	Is x less than y?
x >= y	Is x greater than or equal to y?
x <=y	Is x less than or equal to y?
x == y	Is x equal to y?
x != y	I x not equal to y?

Boolean Expressions and Relational Operators (example)

- Employee is rewarded with a bonus of 500 if his/her sales exceed 5000

```
sales = input('Please enter number of sales')
sales = int(sales)
bonus = 0

if sales>5000:
    bonus = 500

print (f"Bonus Value is {bonus}")
```

Otherwise?

Boolean Expressions and Relational Operators (example)

- You can use multiple statements as part of an if statement. All indented lines after the if statement will be executed if the condition is evaluated to true.

```
sales = input('Please enter number of sales')
sales = int(sales)
bonus = 0

if sales>5000:
    print("Congratulations, you got a bonus!!")
    bonus = 500

print (f"Bonus Value is {bonus}")
```

```
Please enter number of sales 5001
Congratulations, you got a bonus!!
Bonus Value is 500
```

The if-else statement

The `if-else` Statement

- Dual alternative decision structure:

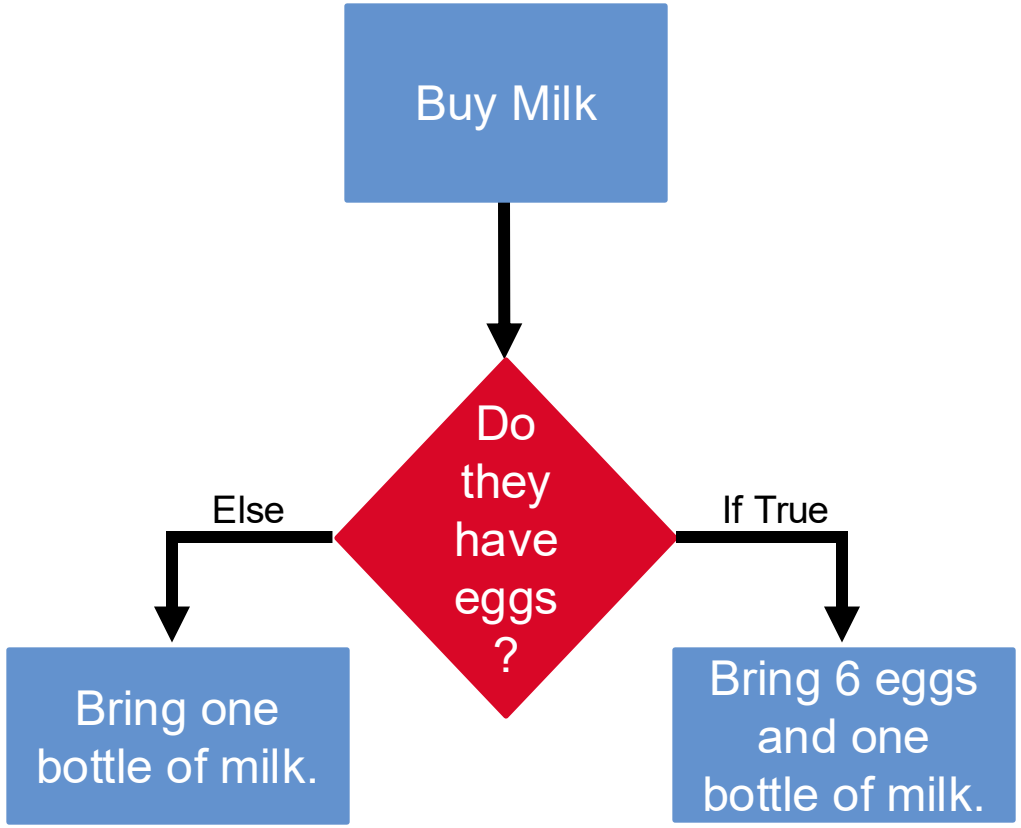
(two alternative paths of execution)

  - One is taken if the condition is true, and the other if the condition is false

  - Syntax:

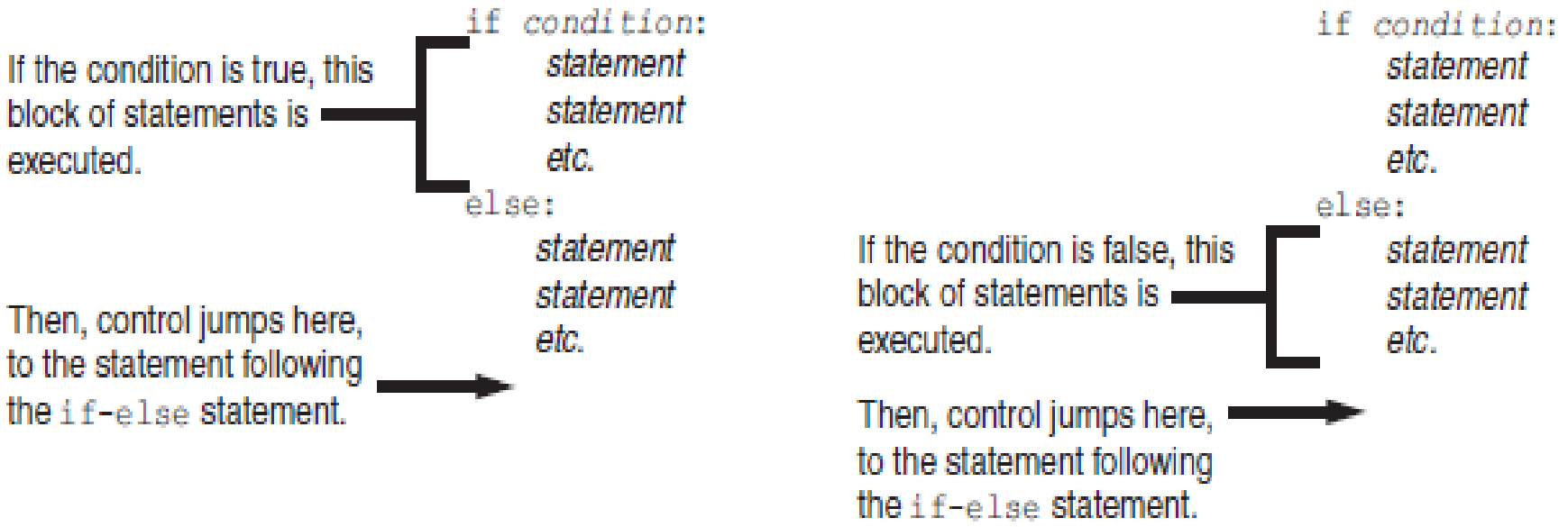
```
if condition:
        statement(s)
else:
        other statement(s)0
```

The `if-else` Statement



The `if-else` Statement (cont'd.)

Conditional execution in an `if-else` statement



The if-else Statement (example)

```
sales = input('Please enter number of sales')
sales = int(sales)
bonus = 0

if sales>5000:
    bonus = 500
else:
    bonus = 100
print ("Bonus Value is ", bonus)
```

Comparing Strings

- Strings can be compared using the == and != operators
  - Just like variables String comparisons are case sensitive

```
name1 = 'Mary'
name2 = 'mary'
if name1 == name2:
    print ("The names are the same")
else:
    print ("The names are NOT the same.")
```

```
The names are NOT the same.
```

Comparing Strings (Convert case)

- Make sure that both strings are comparable.
  - Use the methods .upper() and lower() to change the characters to upper and lowercases.
  - Use the method .title() to transform only the first letter to uppercase.

```
print(f"name1={name1.title()} and name2={name2.title()}")
print(f"name1={name1.upper()} and name2={name2.upper()}")
print(f"name1={name1.lower()} and name2={name2.lower()}")
```

```
name1=Mary and name2=Mary
name1=MARY and name2=MARY
name1=mary and name2=mary
```

```
name1 = 'Mary'
name2 = 'mary'
if name1.title() == name2.title():
    print ("The names are the same")
else:
    print ("The names are NOT the same.")
```

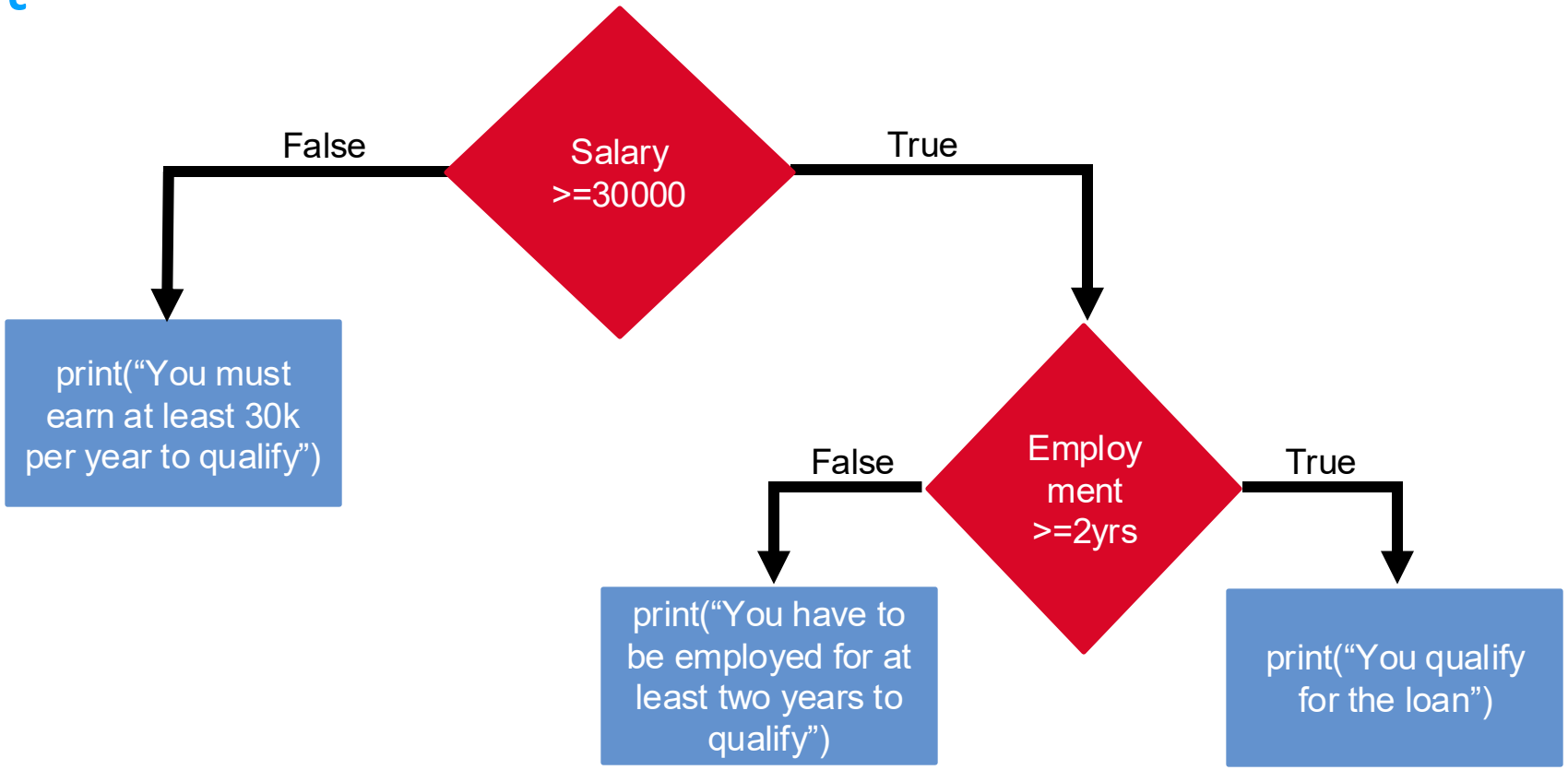
```
The names are the same
```

Nested Decision Structures and the `if-else` Statement

- If you need to evaluate multiple conditions then you can use a nested decision structure (an *if* statement inside another *if*)

  - Example:
    - Determine if someone qualifies for a loan, they must meet two conditions:
      - Must earn at least $30,000/year
      - Must have been employed for at least two years
    - Check first condition, and if it is true, check second condition

Nested Decision Structures and the `if-else` Statement

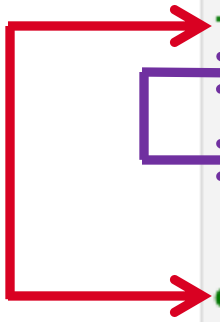


Nested Decision Structures and the `if-else` Statement

- Important to use **proper indentation** in a nested decision structure
  - Important for the Python interpreter
  - Makes the code more readable for the programmer

- Rules for writing nested if statements:
  - **else** clause should align with matching **if** clause
  - Statements in each block must be **consistently indented**

Nested Decision Structures and the `if-else` Statement



```
# Get the customer's annual salary.
salary = int(input("Enter your annual salary:"))
# Get the number of years on the current job.
yearsEmployed = int (input("Enter num years employed"))

if salary >= 30000.0:
    if yearsEmployed >= 2:
        print ("You qualify for the loan.")
    else :
        print ("You must have been on your current")
        print ("job for at least two years to qualify.")
else:
    print ("You must earn at least $30,000 per year")
```

The if-elif-else statement

The `if-elif-else` Statement

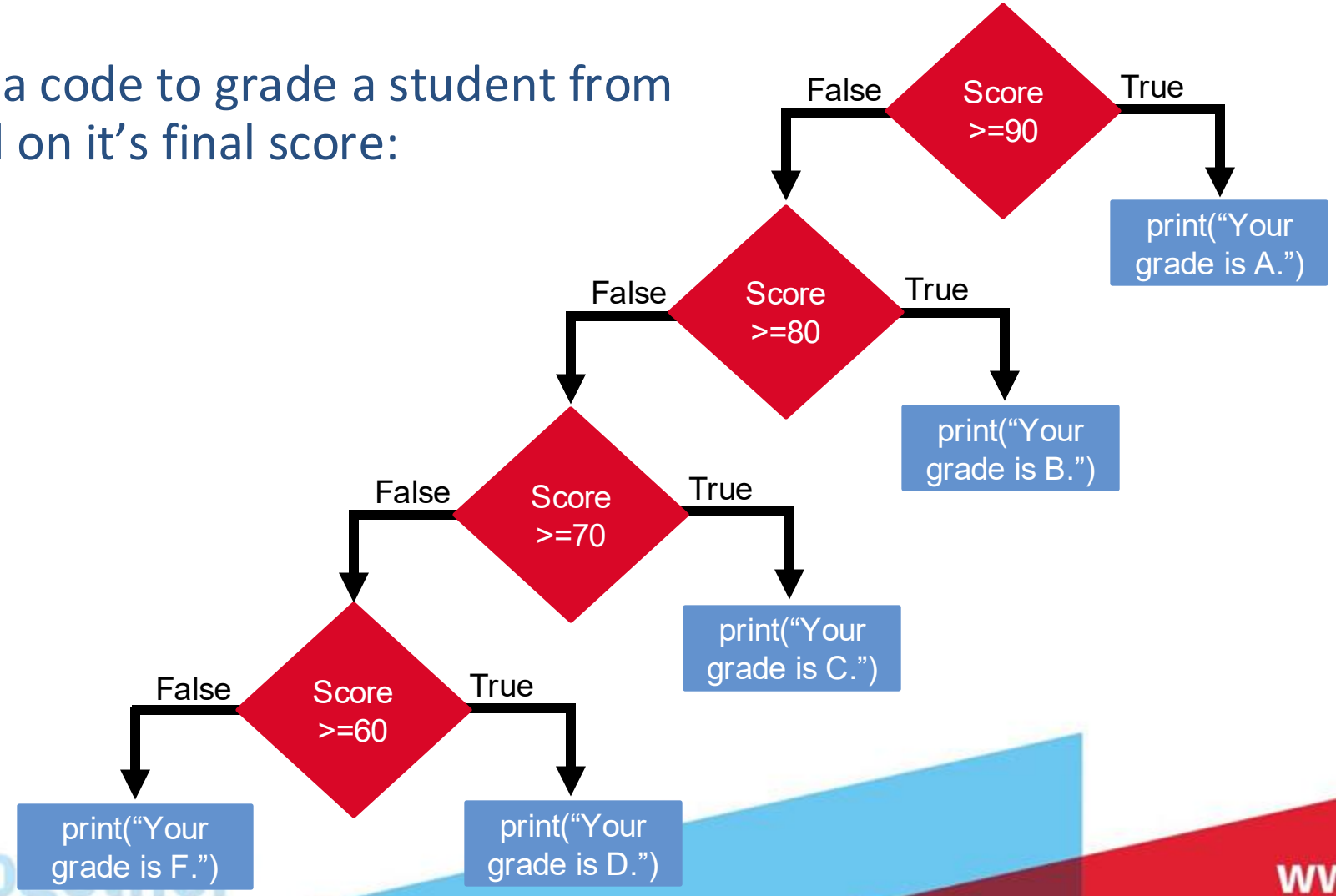
- if-elif-else statement: special version of a decision structure
  - Makes logic of nested decision structures simpler to write
    - Can include multiple elif statements

  - Syntax:

```
if condition1
        statements
elif condition2
        statements
.....
else
        statements
```

Nested Decision Structures and the `if-elif-else` Statement

- Let's create a code to grade a student from A to F based on it's final score:



Nested Decision Structures and the `if-elif-else` Statement

```
# Get the students overall grade.
score = int(input("Enter your exam score:"))
if score >= 90:
    print("Your grade is A.")
elif score >= 80:
    print ("Your grade is B.")
elif score >= 70:
    print ("Your grade is C.")
elif score >= 60:
    print ("Your grade is D.")
else :
    print ("Your grade is F.")
```

elif stands for “Else if”, and its way more memory efficient than nesting else-if statements.

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Logical Operators in decision structures

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
```

Logical Operators in decision structures

- Logical operators: operators that can be used to create complex conditional statements
  - **`and`** operator and **`or`** operator: binary operators, connect two Boolean expressions into a single compound Boolean expression
  - **`not`** operator: unitary operator, reverses the truth of its Boolean operand

The `and` Operator

- Takes two Boolean expressions as operands
  - Creates a compound Boolean expression that is True only when **both sub expressions are true**
  - Can be used to simplify nested decision structures

- Truth table for the **`and`** operator

Expression	Value of the Expression
false and false	false
false and true	false
true and false	false
true and true	true

The and Operator

- The and operator allows us to include more conditions to an if statement.
- Thus, it expands what we can do with it.

```
# Get the students exam score and output the grade.
score = int(input("Enter your exam score:"))

if score >= 60  and score <70:
     print ("Your grade is a D.")
```

The **or** operator

- Takes two Boolean expressions as operands
  - Creates compound Boolean expression that is true when **either of the sub expressions is true**
  - Can be used to simplify nested decision structures

- Truth table for the **or** operator

Expression	Value of the Expression
false and false	false
false and true	true
true and false	true
true and true	true

The **or** operator

- If a user enters a value that is less than 0 or greater than 100 then they should be informed that it is invalid otherwise our code should say invalid.

- Notice that as long as one of the conditions below is true then the subsequent indented code is executed.

```
# Inform use if value received is invalid
grade = int(input(" Please enter exam grade: "))

if grade < 0  or grade > 100:
    print ("Invalid Grade ")
else:
    print ("Valid Grade ")
```

Short-Circuit Evaluation

- Short circuit evaluation: deciding the value of a compound Boolean expression after evaluating only one sub expression
  - Performed by the `or` and `and` operators
    - **For `and` operator**: If left operand is true, evaluate right operand. Otherwise, expression is false.
    - **For `or` operator:** If left operand is true, compound expression is true. Otherwise, evaluate right operand.

The **not** Operator

- Takes one Boolean expressions as operand and reverses its logical value
  - Sometimes it may be necessary to place parentheses around an expression to clarify to what you are applying the not operator
- Truth table for the *not* operator

Expression	Value of the Expression
true	false
false	true

The **not** Operator

- The following code checks the current temperature value and outputs a message it it has not exceeded 30 degrees.

```
temperature = int(input(" Please enter current temperature: "))
if not(temperature > 30):
    print ("This is below the maximum temperature")
```

Boolean Variables in Decision structures

- Boolean variable: references one of two values, `True` or `False`
  - Represented by `bool` data type

```
sales = int(input("Please input total sales"))

if sales >= 50000.0:
    salesQuotaMet = True
else:
    salesQuotaMet = False

if salesQuotaMet == True:
    print ("Congratulations. You will receive a bonus")
```

Boolean conditions are great to create a menu for the user.

match-case statement

A structural pattern matching

- It changes the execution control based on the value of an expression.
- It’s similar to the if-elif-else statement, but it can handle non-binary checks.

```
if action == "coffee”:
    product.append("coffee")
    value += 3
elif action == "tea”:
    product.append("tea")
    value += 2
elif action =="scone":
    product.append(”scone")
    value += 2
else:
    exit()
```

```
match action:
    case "coffee":
        product.append("coffee")
        value += 3
    case "tea”:
        product.append("tea")
        value += 2
    case "scone”:
        product.append(”scone")
        value += 2
    case _:
        exit()
```

Advantages

- Improves the readability of your code and provide a cleaner syntax:

```
if action == "coffee”:
    product.append("coffee")
    value += 3
elif action == "tea”:
    product.append("tea")
    value += 2
elif action =="scone":
    product.append(”scone")
    value += 2
else:
    exit()
```

```
match action:
    case "coffee":
    product.append("coffee")
        value += 3
    case "tea”:
        product.append("tea")
        value += 2
    case "scone”:
        product.append(”scone")
        value += 2
    case _:
        exit()
```

Advantages

- It can be used to match against patterns, not only values

```
match position:
    case (0, 0):
        print("Origin")
    case (0, y):
        print(f"Y={y}") # Matches any position where x is 0
    case (x, 0): # Matches any position where y is 0
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")
    case _:
        print("Not a point")
```

Disadvantages

- It’s an overkill for simple, binary, checks, but it can be used without issues.
- The performance of both methods, case-match or if-elif-else is generally comparable

When to use it then?

When to use it then?

Feature	Use if-elif-else when...	Use match-case when...
Complexity	Conditions are complex (e.g., if x > 5 and y < 10:).	You are matching values or patterns against a single variable or expression.
Python Version	You need compatibility with Python < 3.10.	You can require Python 3.10+.
Use Case	You need simple, universal branching logic.	You are handling complex data structures (tuples, lists, class objects) and their shapes.
Readability	The logic is non-linear or doesn't fit a simple "match this pattern" model.	The logic is a clear list of cases, and the syntax makes it more readable.

How did programmers of the past lived without case-match?

Before Python 3.10

- They did not, they had a workaround:

```
def switch(action):
    if action == "coffee”:
        product.append("coffee")
        value += 3
    elif action == "tea”:
        product.append("tea")
        value += 2
    elif action =="scone":
        product.append(”scone")
        value += 2
    else:
        exit()

print(switch("coffee”))
```

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Python Modules, Packages and Libraries

Python Modules, the power of the community.

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Modules

- Related code saved into a .py file.
- Allow modularity of your program.

Packages

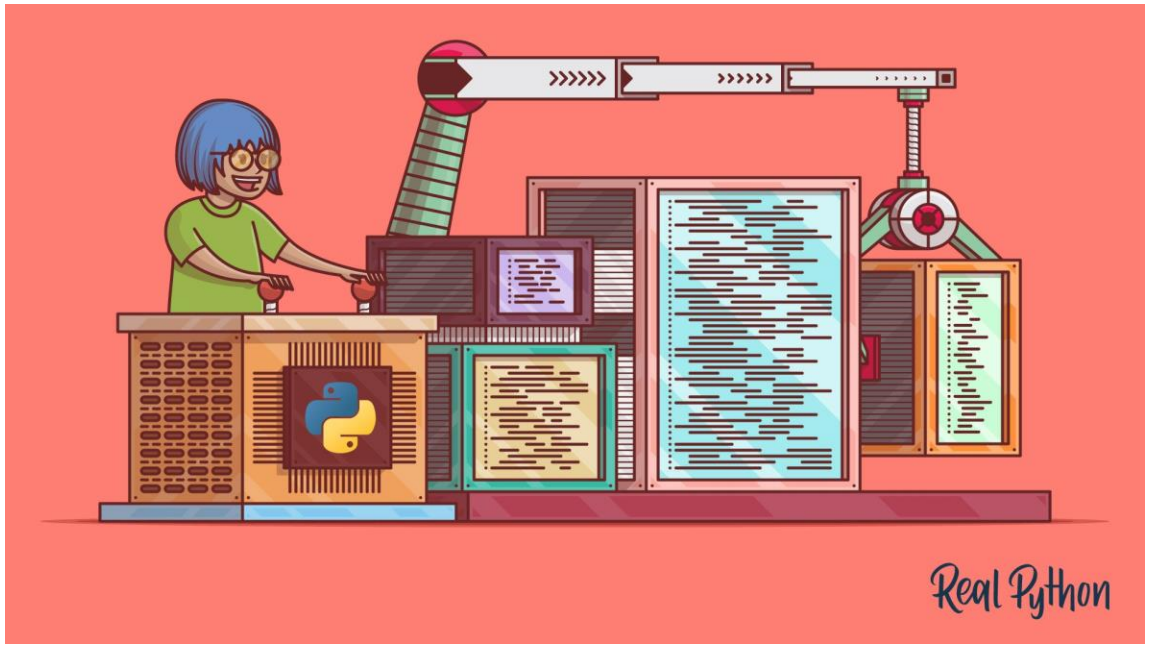
- Are directories of a collection of modules.
- Allow the organisation of related modules.

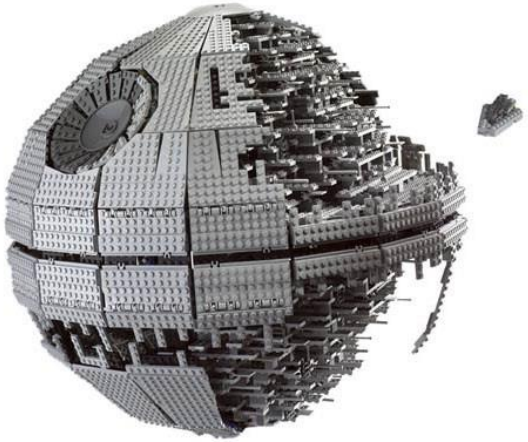
Libraries

- A collection of related Modules and Packages.

Python Modularity, the power of the community.

- Imagine Python as a set of LEGO bricks.
- You have your basic set, with in-built modules.
- But you can expand it with new collections to expand your code functionality!





Why bother coding if you can import code that can do the job?

Examples of Python modules

- Math module.
- Random module.
- Sys module.
- And many more…

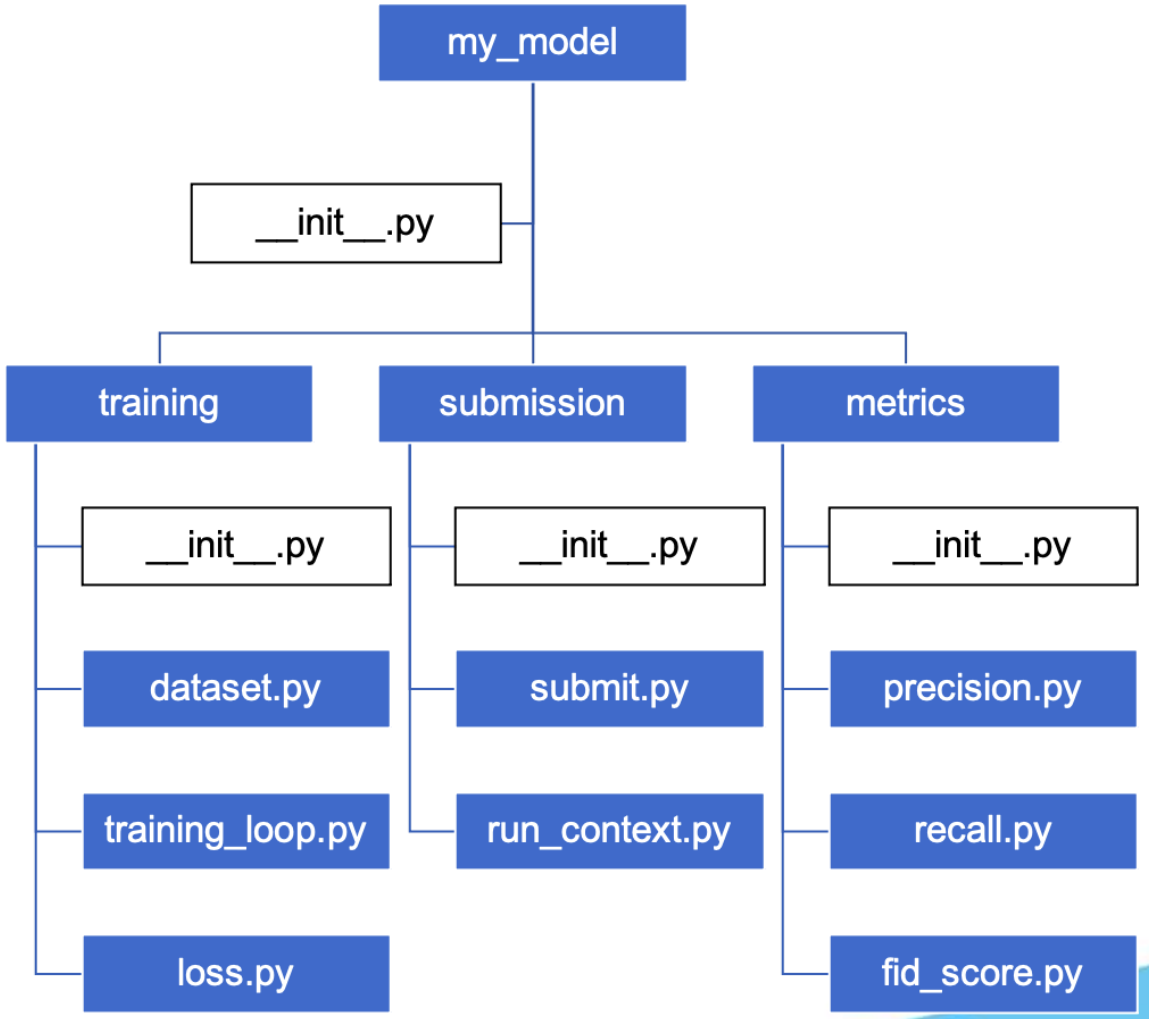
```
def classify_triangle(a: float, b: float, c: float) -> list:
    """
    Determine if a triangle is valid and classify its type.

    Args:
        a, b, c: Three side lengths of a potential triangle

    Returns:
        list: [validity_string, type_string]
    """
```

```
import triangle
```

```
triangle.classify_triangle(3,4,5)
```

```
['Valid', 'Scalene Right']
```

Examples of Packages

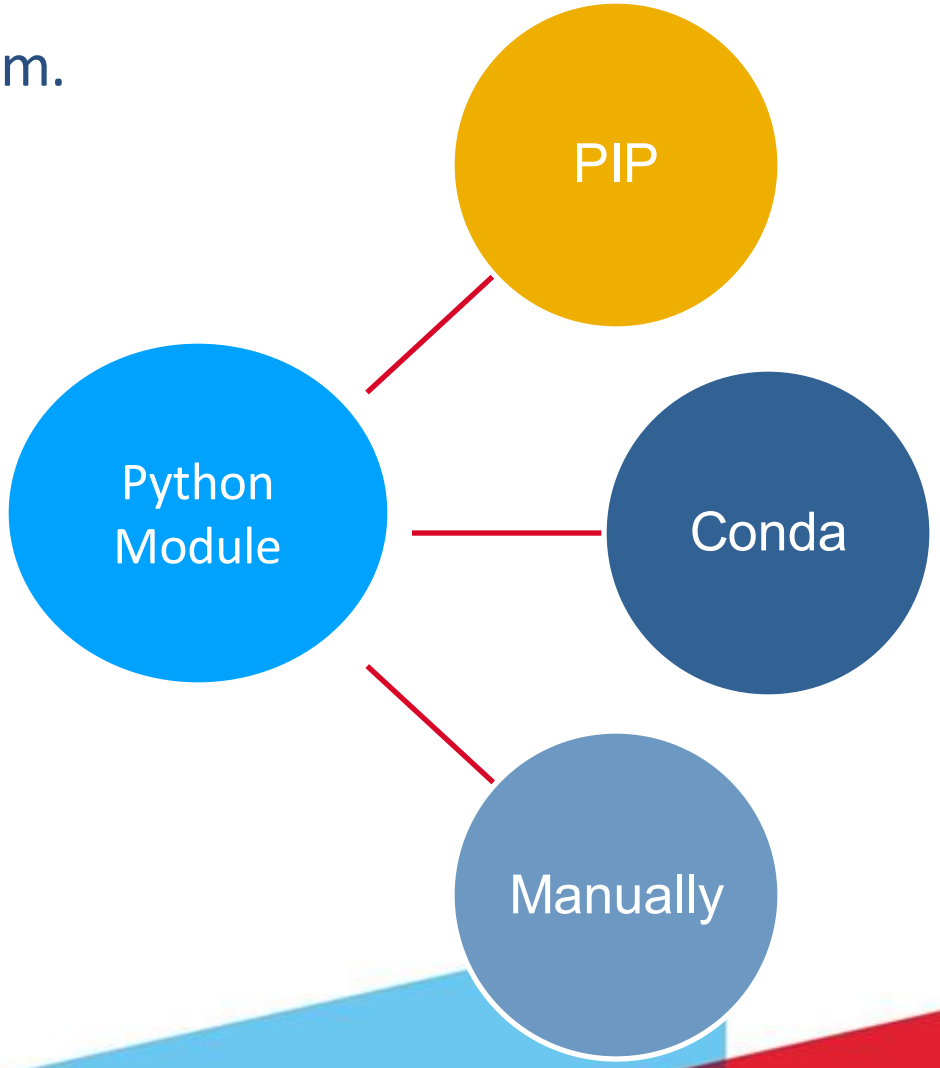


Examples of Packages

Examples of Libraries

Installation.

- There are many ways to install them.



- Let’s check how to do it using PIP.

Pip install.

- Pip is actually a protocol that should be used in an unix terminal, but you can use it on Jupyter or any on other interactive python by adding a ! before pip.

```
!pip install numpy
```

```
Requirement already satisfied: numpy in /User
s/bruno.andrade/miniconda3/lib/python3.8/site-
packages (1.20.3)
```

Import a module/package/library.

- Importing is easy,

```
import math
```

```
import matplotlib
```

```
import numpy
```

```
import numpy as np
```

- Built-in and installed modules will be loaded without any feedback
- You can also give a “nickname” to a module in order to save time, but it’s not the only reason!

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

We give nicknames to libraries to avoid clashes between their and the basic python function names.

Import a module.

- If the module is not installed, an error flag will be raised…

```
import theano
```

```
---------------------------------------------
-----------------------------
ModuleNotFoundError                       Trac
eback (most recent call last)
/var/folders/9x/hx0451rj4n1fcxsm1m6ch5hx7p3m0
h/T/ipykernel_39859/3799233021.py in <module>
----> 1 import theano

ModuleNotFoundError: No module named 'theano'
```

Directly import functions from a library.

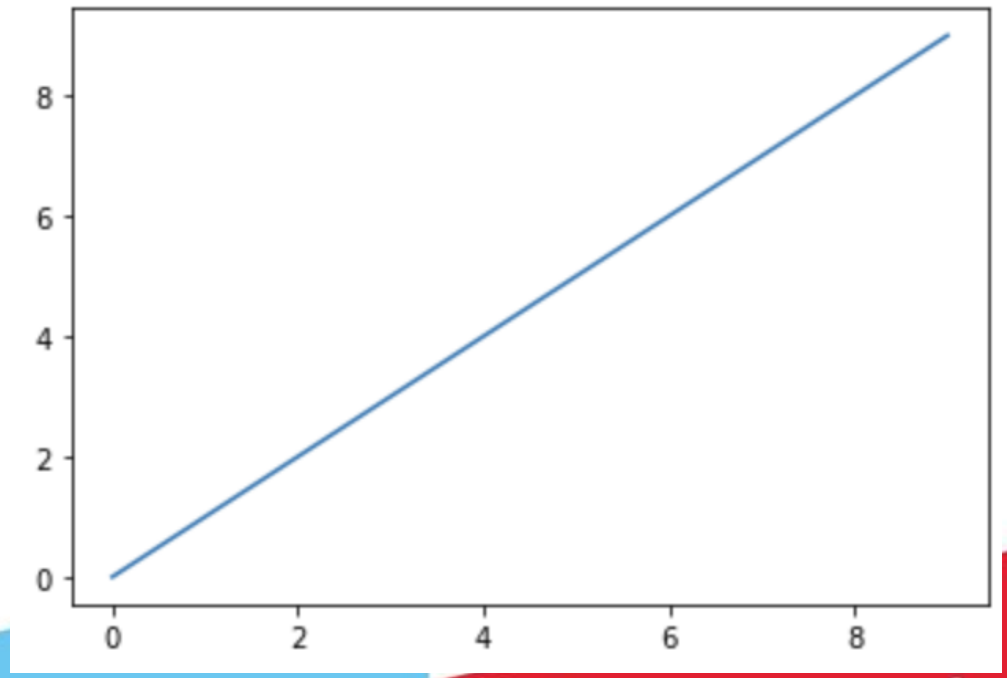
- You can also import modules from a package for the sake of brevity.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Now lets try to create a simple line plot:

```
data = np.arange(10)
plt.plot(data)
```

```
[<matplotlib.lines.Line2D at 0x7f9320c7ac40>]
```



Using methods from a module.

```
import random
```

```
positionX = random.randint(0,100)
print(positionX)
```

```
85
```

```
cakeChoice = random.choice(["Carrot Cake", "Cupcake",
                            "Banana cake","Cheesecake"])
cakeChoice
```

```
'Cupcake'
```

We are going to use the random module a lot.

We will talk more about modules and how to make your own in a couple of weeks.

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

That’s all folks!