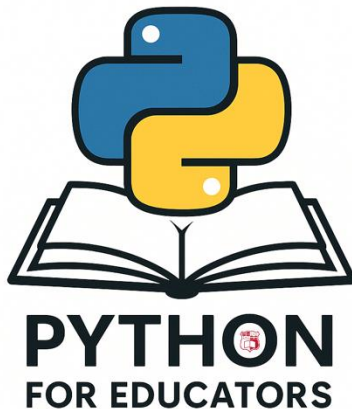


Python 4 Educators



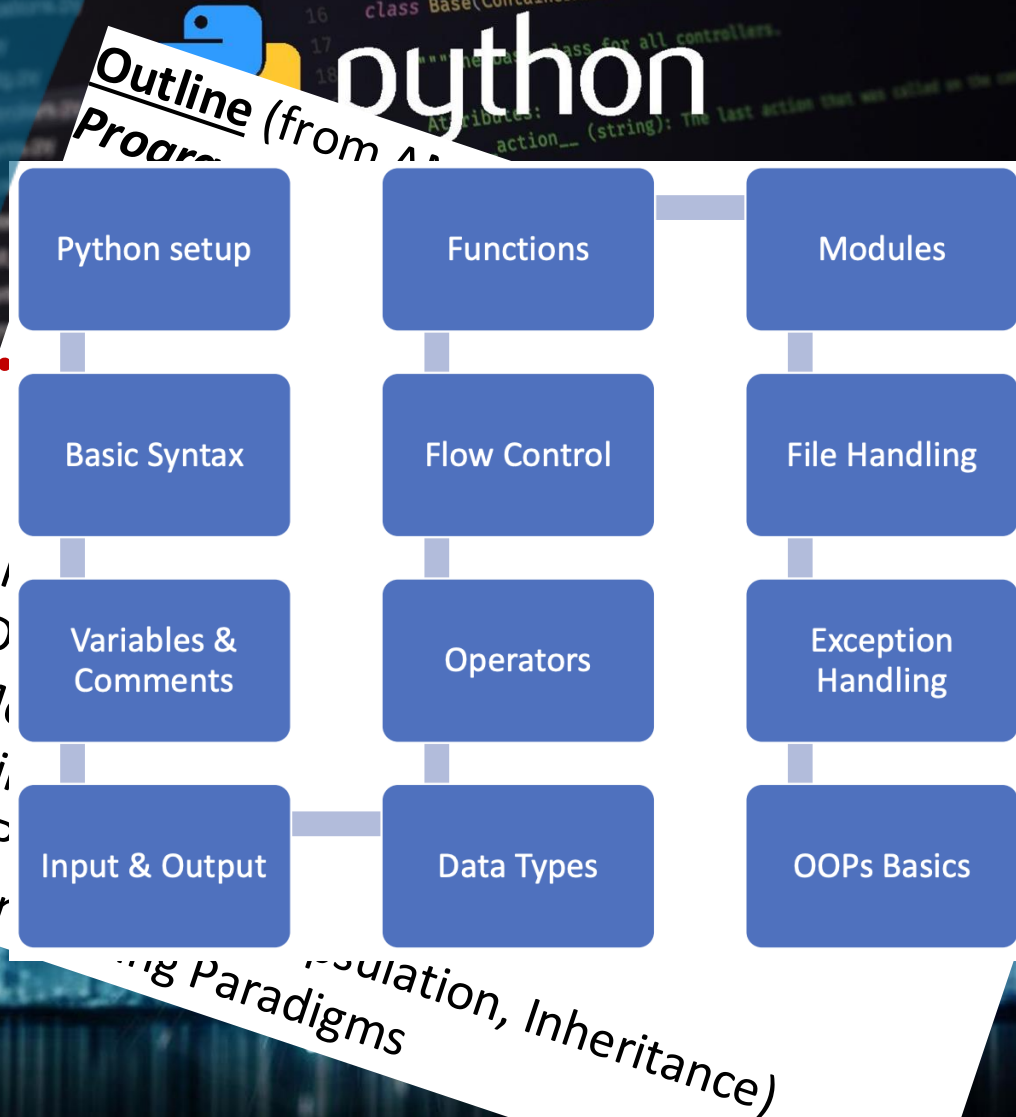
Prof Matthew Montebello



matthew.montebello@um.edu.mt

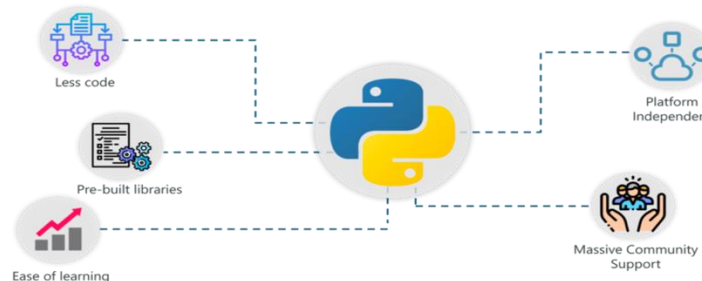


<https://www.um.edu.mt/staff/matthew.montebello>



Welcome

- Python is one of the most popular freely available, versatile, & widely employed coding languages that is interpreted ... not compiled
- Application areas:
 - From server-side web development to data processing;
 - Visualisation as well as App development;
 - Employed to create workflows together with database applications;
 - Fruitfully used to rapid prototype and develop production-ready software.
- Its rise to fame since 1991 since Guido Van Rossum created it in 1989
 - It works on all available platforms as interpreters are freely available for Windows, Mac, Linux, Raspberry Pi, Android, and others.



Welcome

- Python is one of the most popular freely available, versatile, & widely employed coding languages that is interpreted ... not compiled
- GUI Programming – Python supports GUI applications that can be created and ported to many system
- Scalable – Python provides a better structure and support for large programs than shell scripting.
- Portable – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases – Python provides interfaces to all major commercial databases.

Python & Education

Full-stack developer	Software engineer	DevOps engineer	Data analyst
Data scientist	Machine learning engineer	Data engineer	Game developer



- Extensively employed with the manipulation of DATA;
- A need to know, understand and practice the basics of Python;
- Import, clean, manipulate, and visualize data as part of your required skills as a researcher;
- Most popular Python libraries, including Pandas, NumPy, and Matplotlib that allow you to work with real-world datasets.



Python Libraries

- **Seaborn**
 - statistical graphics
- **OpenCV**
 - Image processing
- **Pandas**
 - Data manipulation & analysis
- **StatsModels**
 - Statistical modelling
- **Numpy**
 - Manipulation of multi-dimensional arrays and matrices,

- **Matplotlib**
 - Animated & interactive visualizations
- **Pydot**
 - Graph visualisations
- **Bokeh**
 - Data visualisations
- **Scikit-Learn**
 - Machine Learning - AI
- **Orange**
 - Data visualization, machine learning and data mining toolkit



Programming & Generative AI

- Major shift in Programming education ... educators play crucial role
 - ✓ Collaborating with AI
 - ✓ Thinking critically
 - ✓ Focusing on problem-solving
 - ✓ Promoting responsible AI use
- Employ a handful of GenAI and not just 1
- Look for ideas not answers
- Go through the code ... and:

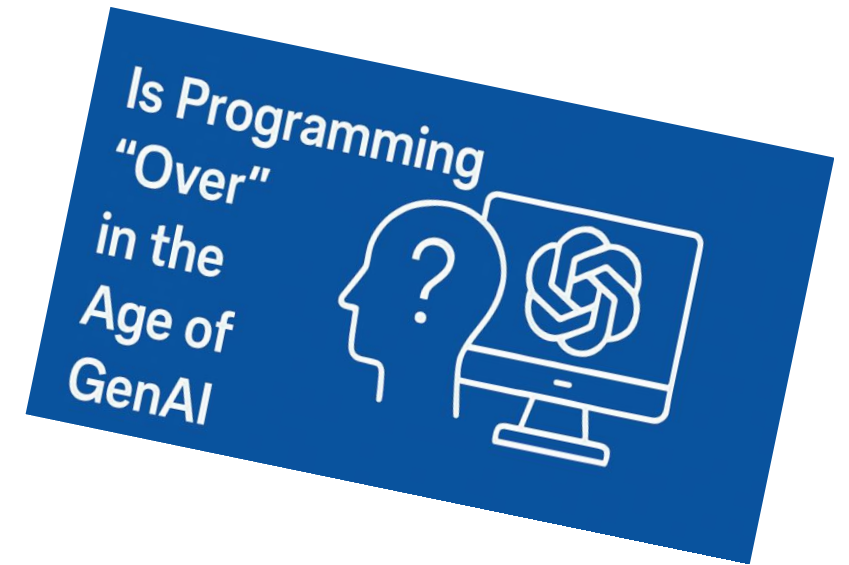
- ✓ Understand it thru & thru
- ✓ Use same GenAI to explain code
- ✓ Same with debugging
- ✓ Adapt & Adopt



Pros & Cons of GenAI in Coding Education

1. Reduced Barriers to Entry

- ✓ *AI tools generate code for beginners*
- ✓ *Less overwhelming for first-timers*
- ✓ *Faster 'success' with simple programs*
- ✗ *Risk of shallow understanding*
- ✗ *Over dependency*
- ✗ *Need for critical evaluation skills*



2. From Syntax to Problem Solving

- *Less focus on memorising syntax*
- *Emphasis on:*
 - *Decomposing problems*
 - *Designing logic*
 - *Debugging AI output*

AI suggests solutions, students evaluate

New skills for Coding with GenAI

3. Prompt Literacy as a Key Skill

- ✓ *Knowing how to ask AI effectively*
- ✓ *Good prompts lead to better code*
- ✓ *Encourages clarity and precision*
- ✓ *Prompt design in programming education*



4. Faster Prototyping and Experimentation

- *AI enables quick testing of ideas*
- *Encourages creative exploration*
- *Ideal for project-based learning*
- *Students can 'fail fast' and iterate*

Final Thoughts related to GenAI

5. The Evolving Role of Educators

Teacher becomes:

- ✓ *A coach for critical thinking*
- ✓ *A guide for debugging and evaluation*
- ✓ *A facilitator for ethics discussions*
- ✓ *Less about syntax, more about understanding*

6. Critical Thinking & Responsible Use

AI suggestions aren't always correct

Students must:

- *Review generated code*
 - *Understand limitations*
 - *Avoid over-reliance*
 - *Discuss ownership*
- & plagiarism concerns*

Session Resources ...



[SIGN UP](#) [LOGIN](#)

Create an Account

GIVEN NAME


SURNAME

EMAIL

PASSWORD

☐ 18 Years or Older

☐ I'm not a robot


reCAPTCHA
[Privacy](#) - [Terms](#)

[CREATE ACCOUNT](#)

To setup an account go to:

www.cgscholar.com

1. Go To Profile - fill in now or later
2. Click on Your Communities
3. Search for the community:

Python 4 Educators

Session 1

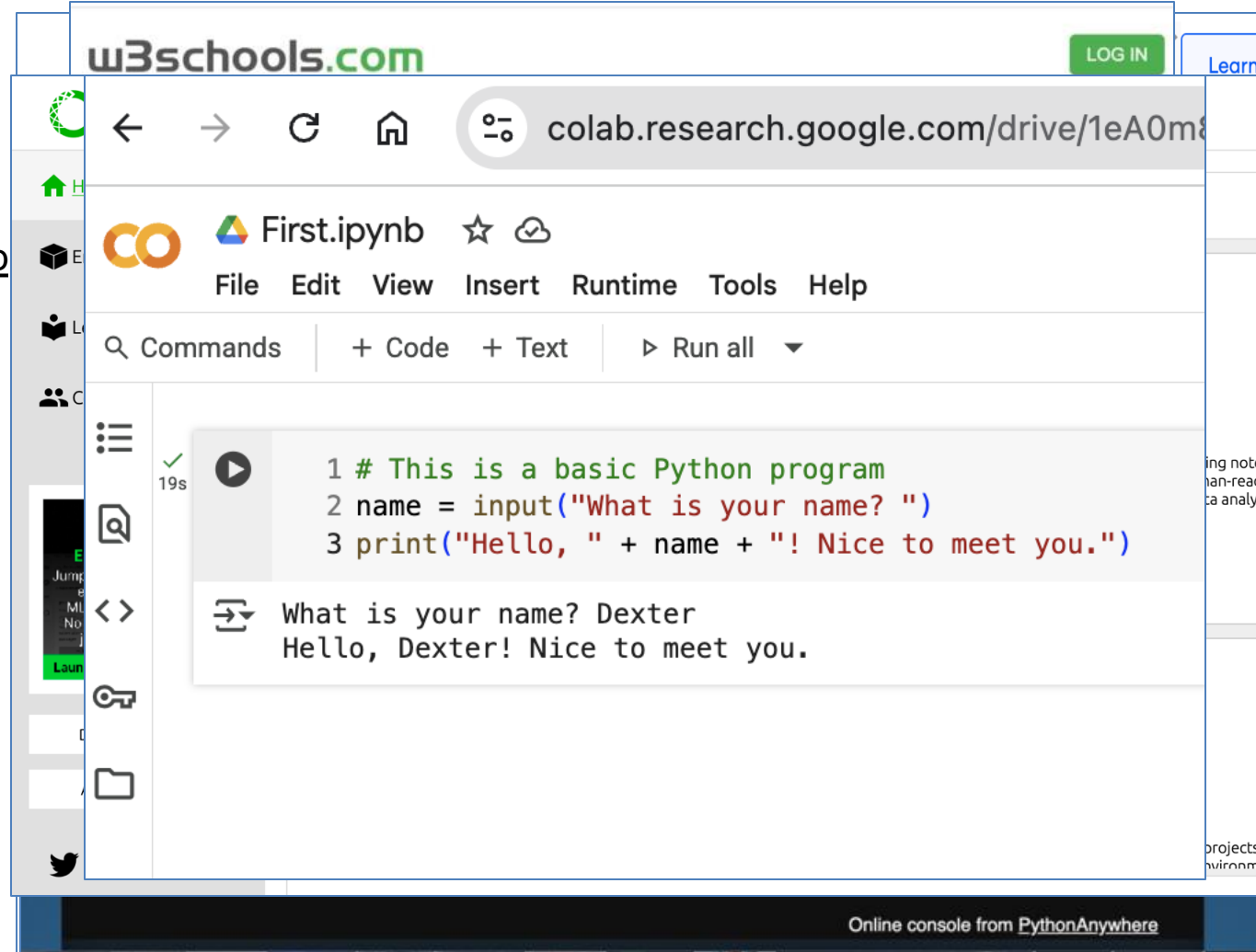
- i. Input and Output
- ii. Variables and Type Conversion
- iii. Arithmetic Operations
- iv. Decision Statements
- v. Iteration
- vi. Programming Errors & Exceptions



Python IDEs

- https://www.w3schools.com/python/python_compiler.asp
- <https://www.programiz.com/python-programming/online-compiler/>
- <https://www.python.org/shell/>
- <https://repl.it/languages/python3>
- https://www.onlinegdb.com/online_python_interpreter
- https://www.tutorialspoint.com/execute_python_online.php
- <https://live.sympy.org/>
- <https://www.pythonanywhere.com/try-ipython/>
- https://rextester.com/l/python3_online_compiler
- <https://ideone.com/>

Other options ...



Some Clarifications for Beginners ...

- Pay particular attention between ...
 - an 'editor' area where you can write/edit code as part of a program that you can then execute ...
 - and the Python console shell itself ... where you can interactively execute commands directly without being part of a Python program.
 - For example in the Shell you do not need to instruct it to print as the Interpreter responds to your query and outputs the result of the execution ... while in the script you have to explicitly instruct the interpreter to output (in this case using 'print') the desired result.
- Tasks will be scattered throughout the course to ensure everyone practices code development in Python ...
 - Code snippets are in the file code#.py on the VLE
 - The icon on the right indicates a task – so get prepared to work out a number of short tasks.



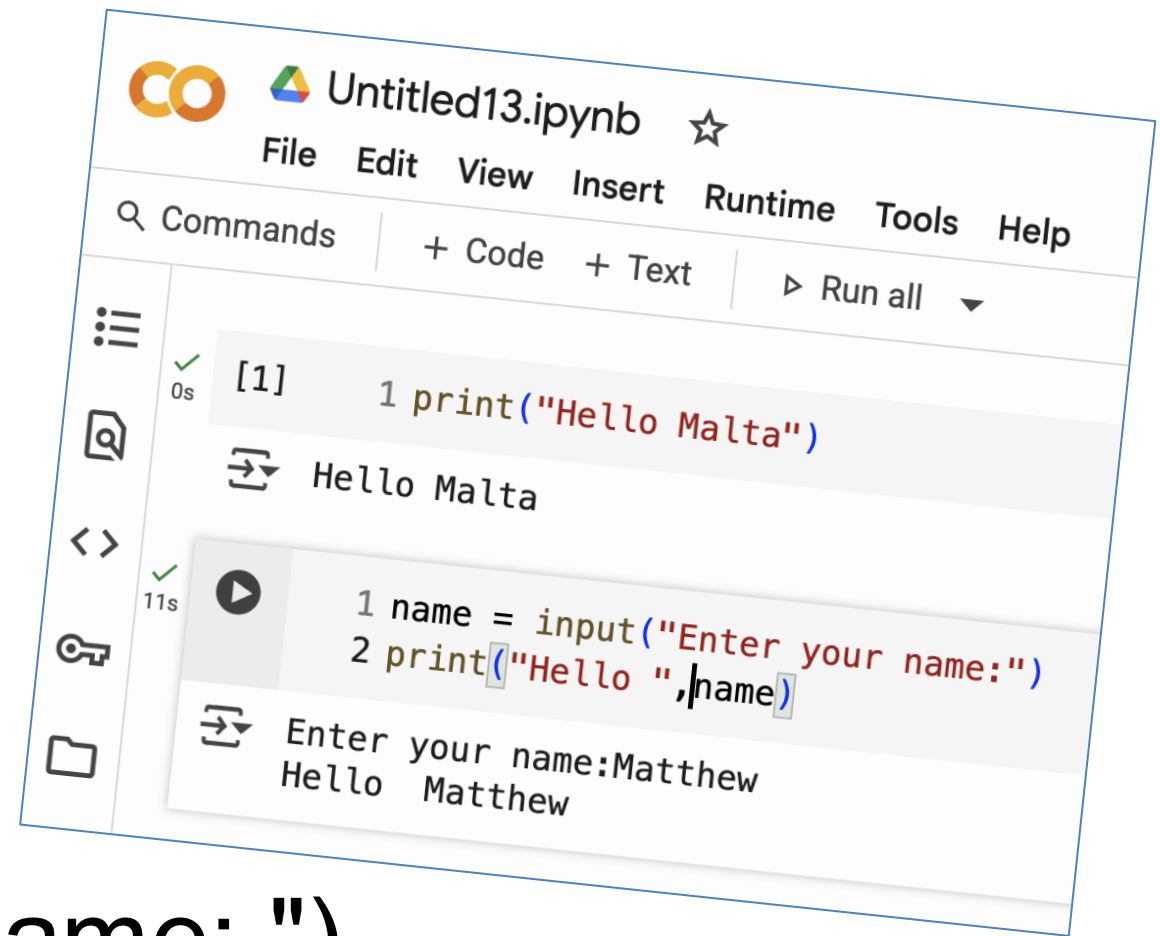
Input and Output

Output Example:

```
print("Hello, World!")
```

Input Example:

```
name = input("Enter your name: ")  
print("Hello,", name)
```



Task 1



- Use the Python Shell ... type **5 + 8**, for example.
- Achieve the same exact result by writing a line of code in the Python script (not in the Shell): **print(5 + 8)** and execute/run the script.
- Try out some other commands ...
 - + Multiplication: Work out the product of 3 and 5
 - / Division: Divide 10 by 2
 - // Floor Division (x // y)
 - % Modulo: What about 18 % 7
 - ** Exponentiation: 4² or 4 ** 2
 - # Comment

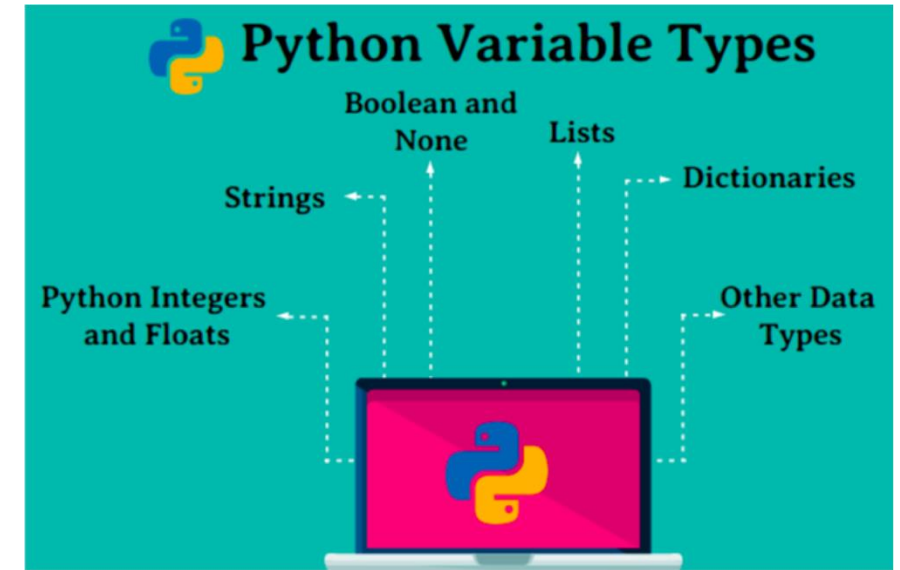
Task 2



- Suppose you invest €1000 with a 10% return each year...
- After one year, it's $1000 \times 1.1 = €1100$
- After 2 years ... it's $1000 \times 1.1 \times 1.1 = €1210$
- How much money would you end up with after 7 years?

Python Variables

- In Python, a variable allows you to refer to a value with a name.
- To create a variable simply use the = sign
- **x = 13 # Variable x has just been created with a value of 13 inside**
- You can now use the name of this variable, x, instead of the actual value, 13.
- Remember: The = sign in Python means assignment, it does not test equality!



Task 3



- Create a variable '**capital**' with the value 1000
- Check that the variable actually contains the Integer 1000

Recall from Task 2 ...investing a €1000 capital for 7 years @ 10% interest

- Create a variable '**interest**' equal to 10% or 0.1
- Create a variable '**result**' equal to the amount of money accumulated after 7 years.
- Print out the value of 'result'.

More Variables ...

- Up to now we worked with 2 variables:
 - **int**, or **integer**: a number without a fractional part.
 - **str**, or **string**: a type to represent text. You can use single or double quotes to build a string.
- Now we will introduce 2 other variables:
 - **float**, or **floating point**: a number that has both an integer and fractional part, separated by a point.
 - **bool**, or **boolean**: a type to represent logical values. Can only be True or False.
- To find out the type of a value or a variable that refers to that value, you can use the ***type()*** function.

Task 4



The same operator can act differently on different types ... for example sum of two integers or floats is different from the sum of two strings (also known as concatenation).

- Practice basic commands:
welcome = 'hello'
user = 'Matt'
message = welcome+user
- Print out the concatenated string
- Check the type of the variable 'result' from Task 3

Converting types ...

- Converting variable types is possible to create a required output string by using the ***str()*** command to convert a value into a string.
- `str(result)`, for example, will convert the float *result* to a string.
- Similar functions help to convert Python values into any type
 - `int()`
 - `float()`
 - `bool()`

- Example:

```
pi_string = "3.1415926"
pi = float(pi_string)
radius = 35

circumference = 2 * pi * radius
```

Task 5



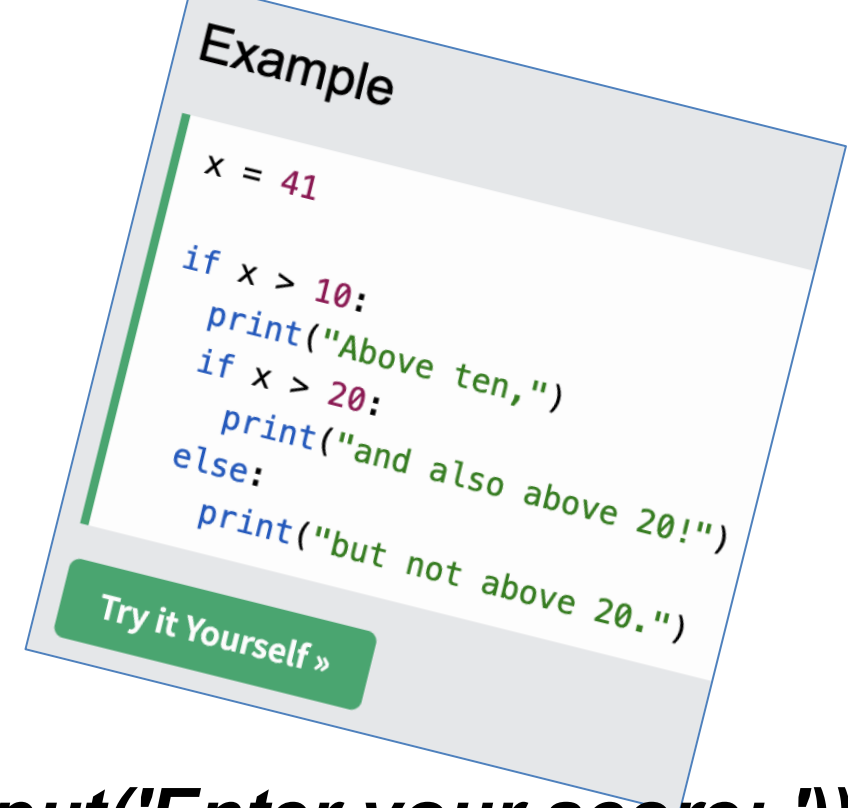
Check out if these are possible or not ...

- "I can add integers, like " + str(5) + " to strings."
- "I said " + ("Hey " * 2) + "Hey!"
- "The correct answer to this multiple choice exercise is answer number " + 2
- True + False
- age = int(input("Enter your age: "))
- height = float(input("Enter your height in meters: "))

Decision Statements (If-Else)

```
age = int(input('Enter your age: '))  
if age >= 18:  
    print('You are an adult.')  
else:  
    print('You are a minor.')
```

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`



```
score = int(input('Enter your score: '))  
if score >= 90:  
    print('Grade: A')  
elif score >= 80:  
    print('Grade: B')  
else:  
    print('Keep improving!')
```

Iteration - While loop & For loop

```
count = 1  
while count <= 5:  
    print('Count:', count)  
    count += 1
```

```
for i in range(1, 6):  
    print('Count:', i)
```

range(start, stop) generates numbers from start to stop-1

Example

Exit the loop when i is 3:

```
i = 1  
while i < 6:  
    print(i)  
    if i == 3:  
        break  
    i += 1
```

Try it Yourself »

```
for x in "banana":  
    print(x)
```

Try it Yourself »

Errors in Python

Syntax Errors: Mistakes in code structure

Runtime Errors: e.g., division by zero

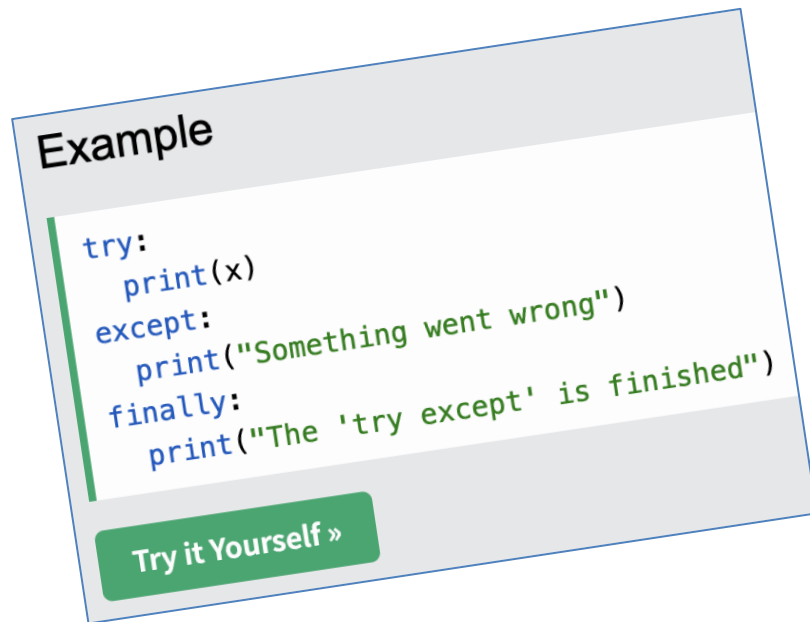
Logical Errors: Code runs but incorrect

Errors	Description
SyntaxError	Occurs when the code violates Python's syntax rules.
NameError	Occurs when you attempt to use or access a variable, function, or module name that has not been defined yet or is out of scope.
IndentationError	Occurs when the spaces at the beginning of a code line do not follow expected patterns.
TypeError	Occurs when an operation is performed or a function is applied on an object of an inappropriate type.
ValueError	Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value.
KeyError	Occurs when a program tries to access a key in a dictionary that doesn't exist.
IndexError	Occurs when you try to access an element in a list, tuple, or any other sequence using an invalid index.
ZeroDivisionError	Raised when you attempt to divide a number by zero.
FloatingPointError	Occur when working with floating-point numbers.
ModuleNotFoundError	Raised when a module could not be found.
ImportError	Raised when an import statement fails to import a module or name within the code.



Exception Handling Example

- The **try** block lets you test a block of code for errors.
- The **except** block lets you handle the error.
- The **else** block lets you execute code when there is no error.
- The **finally** block lets you execute code, regardless of the result of the try- and except blocks.



try:

num = int(input('Enter a number: '))

print(10 / num)

except ZeroDivisionError:

print('Cannot divide by zero!')

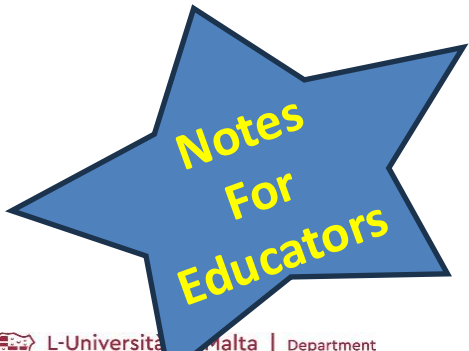
except ValueError:

print('Invalid input! Enter a number.')

Practical Activity - HW

Write a Python program that:

1. Asks the user to enter the number of students in a class.
2. For each student:
 1. Prompts the user to enter the student's name.
 2. Prompts the user to enter the student's test score (0 to 100).
 3. Validates that the score is a number within the correct range. If invalid, prompts again.
3. Calculates and displays:
 1. The average score for the class.
 2. The highest score and the name of the student who achieved it.
4. Displays a **Pass/Fail** message for each student:
 1. A score of 50 or more is a "Pass".
 2. Below 50 is a "Fail".
5. Includes basic error handling to prevent the program from crashing if the user inputs invalid data.



- ✓ Encourages clear input validation with while loops and try/except.
- ✓ Uses floating-point numbers for scores, but integers would also work if preferred.
- ✓ Introduces tracking maximum values and basic arithmetic.
- ✓ Provides immediate feedback with Pass/Fail status.

Practical Activity - Solution

Grading Calculator with Validation and Error Handling

Get number of students

while True:

try:

num_students = int(input("Enter the number of students: "))

if num_students <= 0:

print("Please enter a positive number.")

else:

break

except ValueError:

print("Invalid input. Please enter a valid integer.")

Initialise variables

total_score = 0

highest_score = -1

top_student = ""

Loop over students

for i in range(num_students):

print(f"\nStudent {i+1}")

name = input("Enter student name: ")

Get and validate test score

while True:

try:

score = float(input("Enter test score (0 - 100): "))

if 0 <= score <= 100:

break

else:

print("Score must be between 0 and 100.")

except ValueError:

print("Invalid input. Please enter a numeric value.")

total_score += score

Check for highest score

if score > highest_score:

highest_score = score

top_student = name

Pass/Fail message

if score >= 50:

print(f"{name} has Passed.")

else:

print(f"{name} has Failed.")

Calculate and display average

average = total_score / num_students

print(f"\nClass Average: {average:.2f}")

print(f"Top Student: {top_student} with a score of {highest_score}")

