EMS412U - Python Common Errors Toolkit: *"Squashing Bugs. Not Snakes."*

Muhie Al Haimus **Silvia Valls Santafe** **Dr. Rehan Shah**

January 2026

Please email **m.alhaimus@se23.qmul.ac.uk** for any additional comments or corrections for this toolkit.

Contents

1 Introduction **2**

2 Developing best practices for coding **2**

3 Setting up Python on your own device **2**
- 3.1 Installing Python . . . 2

4 Syntax errors **3**
- 4.1 Spelling mistakes . . . 3
- 4.2 Forgetting to close brackets and quotations . . . 4
- 4.3 Indentation . . . 5
- 4.4 Forgetting to use commas and colons where necessary . . . 6
- 4.5 Interchanging the equality (==) and assignment (=) operators . . . 7
- 4.6 Addressing multiple errors . . . 8

5 Runtime errors **10**
- 5.1 Forgetting to import libraries when needed . . . 10
- 5.2 Dividing by zero . . . 11
- 5.3 Creating infinite loops by forgetting to add a stopping condition (base case) . . . 15
- 5.4 Undeclared variables . . . 16
- 5.5 Concatenating incompatible data types . . . 17
- 5.6 Incompatible Casting . . . 19
- 5.7 Indexing Errors . . . 21

6 JupyterHub specific issues **22**

7 Practice: Find the error and correct it! **23**

8 Answers to error practice questions **25**

9 Dry run template **31**

10 I NEED MORE HELP, WHAT DO I DO? 32
10.1 Bonus: help() built in python function (method) 32
10.2 Written Resources 32
10.3 Video resources 32
10.4 What's next? 32

11 Dry Run Table Answers 33

References 33

1 Introduction

Python is an extremely important skill to master and is used extensively throughout your undergraduate course at Queen Mary University of London. Python is an interpreted programming language, which means that each line of code is translated into machine code and executed (run) one at a time. When the interpreter encounters an error, it stops execution and shows an error message along with the specific line where the issue occurred. On the other hand, compiled languages like C/C++ and Java undergo a compilation process where the entire code is converted into machine code all at once, which can present its own set of challenges for beginners. A great explanation of interpreters is available here (watch up to 1:50) [1].

Understanding how code executes by performing a dry run — where you mentally trace through the code without actually executing it — is a crucial programming skill. This process can be done mentally or by using a provided template (see Section 11). Dry running code helps you examine how the program will behave and identify potential issues before running it.

2 Developing best practices for coding

When learning any new skill it is really important to make sure you don't develop any bad habits. One really important good programming habit is to include comments at the start of your program. These comments **must** include:

- An author
- A date
- A short explanation of the program

Adding comments at the start of your program is helpful for both you and others, as it offers a concise overview of the code's purpose. **In all the subsequent examples you will see this initial commenting style.**

3 Setting up Python on your own device

While not essential for EMS412U, having a local installation of Python can be very beneficial. It enables offline development and gives you the flexibility to install additional libraries or extensions that can enhance your programming experience.

3.1 Installing Python

See the video playlist to help install Python and a Code editor on all platforms here [2] or use the guides as shown below.

Option 1: Install Python interpreter and an Interactive Development Environment (IDE)
Download for Python (Mac, Windows, Linux) [3]
After installing, go into your terminal (for Mac, Linux) or command prompt (for Windows) and type `python3`; this starts the Python interpreter.

On Windows, it automatically installs IDLE which is a very basic IDE for Python.

(Mac, Windows, Linux) IDE's/Code editors to download pick one:
Best choice Visual Studio Code [4] - text editor/IDE
Best IDE Pycharm [5] - you can get a free student licence
Others - Sublime text, Notepad++, and more

Option 2: Install Anaconda distribution
Anaconda distribution is a powerful platform for data science, machine learning, and AI projects. It includes a Python interpreter, thousands of open-source packages, like numpy and Jupyter Notebook, a package environment manager, and also a good IDE called Spyder.

You can easily download it here. [6]

A video to help you install Anaconda can be found on the QMPlus Python IT classes-JupyterHub section.

4 Syntax errors

Errors can look really scary at first, but don't panic as almost all errors are really easy to fix!
Syntax errors are very easy to fix! You might now be wondering what exactly the word 'syntax' means. At the most basic level, syntax are keywords and operators of any programming language. These **never** change so it is really important to make sure that you stick to the rules of using syntax. Some common syntax errors include:

4.1 Spelling mistakes

Explanation: As you can see below, `print` is spelled incorrectly. This causes the program to generate a syntax error. This is the most basic programming mistake you can make.

```
# Author: Muhie
# Date: 21/05/24
# Explanation of program: of Outputs Hello from Queen Mary University of London! to the screen
pint("Hello from Queen Mary University of London!")
```

Figure 1: A program with a spelling mistake that causes the program to crash

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 4
      1 # Muhie
      2 # 21/05/24
      3 # Outputs Hello from Queen Mary University of London! to the screen
----> 4 pint("Hello from Queen Mary University of London!")

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

Figure 2: Error output showing the syntax error that `pint` is not defined

Solution: All you need to do is fix the typo `pint` to `print` and the program will run correctly.

Tip: *In Python, words like `print`, `sum`, `len`, `input`, `range`, `int`, `float`, or `str` are keywords and built-in functions. If you misspell them, your code will not work.*

4.2 Forgetting to close brackets and quotations

Explanation: The program below adds two variables `a` and `b` together. However, the print statement contains a syntax error as the opening bracket has not been closed.

```
# Author: Muhie
# Date: 21/05/24
# Explanation of program: adds the variables a and b together
a = 2
b = 4
print(a + b
```

Figure 3: A program that has incomplete brackets which causes the program to crash

```
  Cell In[2], line 6
    print(a + b
               ^
SyntaxError: incomplete input
```

Figure 4: Error output showing how the `print` statement is incomplete

Solution: All that is required to resolve the error here is to close the brackets after `a + b`.

Explanation: I have now made a new program. That stores the name of the user in the variable `name`. However, the program does not run as the quotation mark is missing at the end of the `input` statement.

```
# Author: Muhie
# Date: 21/05/24
""" Explanation of program: Asks for and stores the name of users
in the variable name and then outputs it to the screen """
name = input("What is your name?)
print(name)
```

Figure 5: A program with incomplete quotation marks that causes the program to crash

```
  Cell In[3], line 5
    name = input("What is your name?)
                 ^
SyntaxError: unterminated string literal (detected at line 5)
```

Figure 6: Error output showing how the string has not been terminated and hence not completed

Solution: A closing quotation mark is required at the end of the `input` statement.

Key point: It is really important to make it a habit to add an accompanying closed bracket or quotation mark whenever you open a bracket or quotation mark to eliminate this trivial error. *This applies to brackets (), quotation marks "", square brackets [] and curly braces* {}. *Forgetting to close any of these will cause a syntax error.*

4.3 Indentation

Explanation: Indentation shows which lines of code belong together. In Python, it is critical to have the correct indentation to avoid basic errors. One indent is either one press of the tab key or four presses of the space bar. In Python, indentation is used to show where a new block of code is contained and this must be proceeded by a colon as shown in Figure 7, in which the `print` statement is treated as being of the `if` block of code.

```
# Author: Muhie
# Date: 21/05/24
# Explanation of program: checks if 5 is greater than 2
if 5 > 2:
    print("Yes it is duh")
```

Figure 7: A program that shows the correct application of a colon followed by indent

Output:
`Yes, it is Duh`
Be careful about the indentation, do not indent your code in the wrong level, since this may not trigger an error, but cause your program to give unexpected results.

Explanation: Here is another example that demonstrates how the wrong indentation can cause unexpected

outputs.

```
# Author: Muhie
# Date: 21/05/24
# Explanation of the program: print the multiplication table
# at the size of table_size. For example,
# A multiplication table at a size of 3 is :
# 1  2   3
# 2  4   6
# 3  6   9

table_size = 3

for i in range(1, table_size + 1):
    for j in range(1, table_size + 1):
        value=i*j
    print(str(value),end="   ")    # This line is indented at a wrong level.
    #You should add an indentation to fix this.
    print()  # Newline after each row
```

Figure 8: A program that demonstrates the consequences of incorrect indentation

Output:
3
6
9

Solution: In this example, the program runs without any errors. Since the print statement has one indentation less, the program only prints the last column of the multiplication table.

4.4 Forgetting to use commas and colons where necessary

This is a common mistake, however it is usually easy to spot and correct. These mistakes are most likely to occur when creating loops, defining specific functions, or creating conditional statements. Here you can see an example containing various such mistakes.

Explanation: This program shows the implications of forgetting to use a colon after the declaration of a `for` statement, leading to an error output.

```
#Author: Yash Vaghela
#Date: 05/08/2024
'''Explanation of the program: The defined function returns the output of
the printed statement within the for loop, in this case it outputs 3 "1"s'''

def loop(num1, num2):
    for i in range(num2)
        print(num1)
    return
loop(1, 3)
```

Figure 9: A code snippet showing a missing colon after the `for` loop declaration

```
  Cell In[3], line 6
    for i in range(num2)
                        ^
SyntaxError: expected ':'
```

Figure 10: Output showing the syntax error in line 6

Solution: Usually, the output error will highlight if something is missing, so it would be easy to rectify the mistake. In this case you would need to add a '`:`' after the `for` loop declaration.

Tip: *Always add a colon (:) after statements like `if`, `for`, `while`, `def`, and `else`.*

4.5 Interchanging the equality (==) and assignment (=) operators

In Python, operators are categorised into several groups, including arithmetic (+, −, *, etc.), comparison (>, <=, !=, etc.), and logical (`and`, `or`, `not`). A double equal sign (==) is an equality comparison operator that checks if two values are equal and returns either `True` or `False`. A single equals sign (=) is an assignment operator used to assign a value to a variable. These operators are distinct and cannot be used interchangeably. In other words, = != ==.
For instance, consider this short program that verifies whether the user-entered password (stored in a variable called `password`) matches the value in `correct_password` and displays a corresponding message.

```python
# Author: Ilan
# Date: 03/08/2024
'''The program checks if the user's input matches a
predefined password and grants access if it does,
or denies it otherwise.'''

correct_password = "theDukeOfEelam30!"
password = input("Enter password: ")
# Mistaken use of '=' instead of '=='
if password = correct_password:
    print("Welcome Ilan!")
else:
    print("Password not recognised.")
```

Figure 11: Code snippet with a mistake using = instead of ==

Explanation: In the above program, the incorrect use of the assignment operator within the condition of the selection statement results in the following syntax error.

```
  Cell In[1], line 10
    if password = correct_password:
       ^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

Figure 12: Cell output showing the syntax error on line 10

Solution: All you need to do is navigate to line 10 and replace the assignment operator with the comparison operator ==.

Be careful! *Sometimes using '=' instead of '==' will not cause an error, it might just make your program behave incorrectly, which can be harder to spot.*

4.6 Addressing multiple errors

In programs with multiple errors, only the first error is typically shown during the initial run. This error prevents the program from executing further, but it does not imply that there is just one issue. After fixing the first error, running the program again usually uncovers additional errors that were previously hidden.

```python
# Author: Ilan
# Date: 28/08/2024
'''The program is intended to print a birthday message
when a name and age are passed into a procedure.'''

def birthdayWish(name, age_now):
print("Happy Birthday,", name)
    print("You are turning", age_now, "today. Exciting!")

name, age_now = "Krishni" 20
birthdayWish(name, age_now)
```

Figure 13: Code snippet of a program with an indentation and syntax error

Explanation: Here, there are two errors: first, the line following the function definition is not properly indented, and second, there is a missing comma in the assignment of the `name` and `age_now` variables. However, only the first error is identified during the initial run of the program, as shown below.

```
  Cell In[29], line 7
    print("Happy Birthday,", name)
    ^
IndentationError: expected an indented block after function definition on line 6
```

Figure 14: Cell output showing the indentation error on line 7

Once this line is correctly indented and the program is run, the cell output will then highlight the syntax error in the variable assignment.

```
  Cell In[30], line 10
    name, age_now = "Krishni" 20
                              ^
SyntaxError: invalid syntax
```

Figure 15: Cell output showing the syntax error on line 10

Solution: The error is caused by the missing comma between the string `Krishni` and the integer `20`. Adding a comma will correct the variable definition. The corrected program and its outputs are shown below.

```python
# Author: Ilan
# Date: 28/08/2024
'''The program is intended to print a birthday message
when a name and age are passed into a procedure.'''

def birthdayWish(name, age_now):
    print("Happy Birthday,", name)
    print("You are turning", age_now, "today. Exciting!")

name, age_now = "Krishni", 20
birthdayWish(name, age_now)
```

```
Happy Birthday, Krishni
You are turning 20 today. Exciting!
```

Figure 16: Code snippet of the corrected program, along with the output produced

5 Runtime errors

Unlike syntax errors, which occur before the program actually runs, runtime errors happen during the execution of the program. Although the program will run, it can sometimes crash or just not work as expected. This could be a mistake as simple as using the wrong mathematical operator like using a (<) symbol in place of a (>) symbol. Sometimes runtime errors can be very hard to debug so it is very useful to have some strategies in place to speed up the debugging process.

5.1 Forgetting to import libraries when needed

When solving specific problems using Python, in most cases, you need to use certain libraries since not every process/function you may want will be directly available to you. However, a common mistake by users is they assume the functions specific to certain libraries are automatically available. Therefore, it is good practice to identify all the libraries you may require for your work and import all of them (or the specific functions from the libraries) at the beginning of your Python code. An example of some code where the library was not imported correctly:

Explanation: The code below shows an error due to the user importing the required functions from the math library after calling them rather than before, leading to Python not being able to identify the `sin` and `pi` function.

```python
'''Author: Yash
   Date: 03/08/24
   Explanation of the program: it should print to show sin(pi/6) = 0.5'''
print(sin(pi / 6))
from math import sin, pi
```

Figure 17: Code snippet showing incorrect order of declaration of the imported functions

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 4
      1 '''Author: Yash
      2    Date: 03/08/24
      3    Explanation of the program: it should print to show sin(pi/6) = 0.5'''
----> 4 print(sin(pi / 6))
      5 from math import sin, pi

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

Figure 18: An error output showing that the `sin` function is not identified

Solution: The course of action here would be to move the print statement after 'line 5'. Doing so, imports the `sin` and `pi` functions from the math library, allowing the code to run.

Remember: Python only loads what you tell it to. Built-in functions like `print()` work automatically, but anything from extra libraries, such as `math`, `random`, `datetime`, `pandas`, `numpy`, or `matplotlib`, must be explicitly imported.

5.2 Dividing by zero

This is probably the easiest runtime error to detect as it is $always$ causes the program to crash as, obviously, it is impossible to divide anything by zero. [7]

Explanation: One common place where a division by zero error may occur is when using for loops (counter-controlled loops). When manipulating the counter that increments/decrements, sometimes, the counter may become zero due to applying a mathematical operation to it. For example in Figure 19, when the counter `i`, reaches 5 the denominator of the fraction becomes zero, causing the program to crash.

```
# Author: Muhie
# Date: 5/09/24
# Explanation of the program: A program that finds 6/(x**2-5)
# between 1 and 5

x_values = []
y_values = []
for x in range(1, 6):
    x_values.append(x)
    y = 6/((x**2)-25)
    y_values.append(y)

print("The values of x between 1 and 5 are: {}".format(x_values))
print("The values of y between 1 and 5 are using the \n" +
"function f(x)=6/((x**2)-5) are: \n{}".format(y_values))
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[17], line 9
      7 for x in range(1, 6):
      8     x_values.append(x)
----> 9     y = 6/((x**2)-25)
     10     y_values.append(y)
     12 print("The values of x between 1 and 5 are: {}".format(x_values))

ZeroDivisionError: division by zero
```

Figure 19: A faulty program that calculates $\frac{6}{x^2-5}$ between 1-5, and crashes on the fifth iteration

Solution: Division by zero is certainly not possible (thanks to the world of mathematics). Although, there are ways to prevent the program from crashing entirely. The best solution to this problem is to use an `if statement` to catch if the counter ever becomes zero and then skip the division operation on that iteration.

```
# Author: Muhie
# Date: 5/09/24
# Explanation of the program: A program that finds 6/(x**2-5)
# between 1 and 5

x_values = []
y_values = []
for x in range(1, 6):
    x_values.append(x)
    if ((x**2)-25) != 0: # check that no division by zero error can occur
        y = 6/((x**2)-25)
        y_values.append(y)
    else:
        print("y is undefined at x = {}!".format(i))

print("The values of x between 1 and 5 are: {}".format(x_values))
print("The values of y between 1 and 5 are using the \n" +
"function f(x)=6/((x**2)-5) are: \n{}".format(y_values))
```

```
y is undefined at x = 5!
The values of x between 1 and 5 are: [1, 2, 3, 4, 5]
The values of y between 1 and 5 are using the
function f(x)=6/((x**2)-5) are:
[-0.25, -0.2857142857142857, -0.375, -0.6666666666666666]
```

Figure 20: Shows how a total program crash can be prevented with a `if statement`

Explanation: Another place a division by zero error occurs regularly is when handling a user's input. The program below has a division by zero error due to the user's input.

```
# Author: Muhie
# Date: 5/09/24
# Explanation of the program: A program that finds 6/(x**2-5)
# between 1 and 5


numerator = int(input("Welcome to the fraction calculator, please enter the numerator "))
denomenator = int(input("Please enter the denomenator "))
fraction = numerator/denomenator
print(fraction)
```

```
Welcome to the fraction calculator, please enter the numerator  10
Please enter the denomenator  0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[33], line 9
      7 numerator = int(input("Welcome to the fraction calculator, please enter the numerator "))
      8 denomenator = int(input("Please enter the denomenator "))
----> 9 fraction = numerator/denomenator
     10 print(fraction)

ZeroDivisionError: division by zero
```

Figure 21: A program that shows how a users' input can cause it to crash

Solution: The easiest way to deal with these errors is to add input validation. This can be done using a `while` loop and an `if` statement.

```python
# Author: Muhie
# Date: 21/05/24
# Explanation of the program: A program that finds fractions
def get_valid_input(message, zero_check):
    valid_input = False
    while valid_input == False:
        user_input = int(input(message))
        if zero_check == False:
            return user_input
        if  user_input != 0:
            valid_input = True
            return user_input
        print("Invalid input, you cannot divide by zero, plese try again")

numerator = get_valid_input("Welcome to the fraction calculator, " +
                                  "please enter the numerator", False)
denomenator = get_valid_input("Please enter the denomenator", True)
fraction = numerator/denomenator
print(fraction)
```

```
Welcome to the fraction calculator, please enter the numerator 10
Please enter the denomenator 0
Invalid input, you cannot divide by zero, plese try again
Please enter the denomenator 0
Invalid input, you cannot divide by zero, plese try again
Please enter the denomenator 0
Invalid input, you cannot divide by zero, plese try again
Please enter the denomenator 8
1.25
```

Figure 22: Code with input validation

5.3 Creating infinite loops by forgetting to add a stopping condition (base case)

The simplest case where a user could accidentally create an infinite loop is through the use of the ‘`while`’ loop functions. An example case consists of using ‘`while True:`’, this statement in conjunction with the boolean `True`, makes it so the `while` loop runs infinitely until the loop is manually broken by the user. As shown below, a `while` statement and the condition $a < b$ (which is always true), makes the loop run infinitely, since there is never a condition that causes the loop to break. For example:

```
#Author: Yash Vaghela
#Date: 04/08/2024
'''Explanation of the program: It will print "0" infinitely until the browser crashes.'''

a = 0
b = 3
while a < b:

    print(a)
```

Figure 23: Simple infinite loop case

The infinite loop above can be terminated by simply adding '$a = a + 1$' before the `print(a)` line within the while statement. This makes it so that $a < b$ is only true for 3 loops, hence the loop will end up printing the numbers 1, 2 and 3. It can be seen that infinite loops can arise from setting inappropriate conditions on '`while`' loops, and failing to update loop variables or setting incorrect variables can also cause issues. Therefore, double-checking your code confirming it makes sense, and no conflicting or improper conditions exist within your code.

Tip: *When testing loops, it is safer to start with a small range or include a temporary* `print()` *or* `break` *condition, so you can see what is happening before it runs too long.*

5.4 Undeclared variables

Variables are named memory locations used to store data. Setting up a variable is rather simple; for example, `institution = "QMUL"` assigns the text "QMUL" to the variable named `institution`. In Python, variables must be assigned a value before they can be used. Using an undeclared variable will result in a `NameError`. For example, in the following code snippet, the variable named `city` is being called for but not yet defined!

```
# Author: Ilan
# Date: 04/08/2024
'''The program is intended to randomly select a city from
an array of cities, and is to output this random choice.'''

import random
my_cities = ["Jaffna","London","Chennai","Tokyo"]
random.choice(my_cities)
print(city)
```

Figure 24: Code snippet with an error due to the undefined variable `city`

Explanation: The error occurs because the `print(city)` line references a variable named `city`, which has not been defined earlier in the code. The previous line randomly selects an element from the `my_cities` list, but the selection is not stored in a variable. It's evident that the intention was to store this random choice in the `city` variable. The following is the error traceback.

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 9
      7 my_cities = ["Jaffna","London","Chennai","Tokyo"]
      8 random.choice(my_cities)
----> 9 print(city)

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

Figure 25: Python traceback showing a `NameError` due to the undefined variable reference on line 9

Solution: To fix this, assign the random choice to the `city` variable before the `print` statement. In the following figure, the random selection from `my_cities` is correctly assigned to `city`, allowing the program to run without errors (in this case, producing the output "Jaffna").

```
# Author: Ilan
# Date: 04/08/2024
'''The program is intended to randomly select a city from
an array of cities, and is to output this random choice.'''

import random
my_cities = ["Jaffna","London","Chennai","Tokyo"]
city = random.choice(my_cities)
print(city)
```

```
Jaffna
```

Figure 26: Code snippet of the corrected program, along with the output produced

Note: It's important to follow naming conventions for variables and avoid using Python keywords (e.g., `print`) as identifiers, as this can lead to unexpected behavior and errors in your code. Python can only use a variable after it has been created in the current scope. If a variable is defined inside a function or loop, it won't be accessible outside of it unless it's returned or declared globally.

5.5 Concatenating incompatible data types

Data types define the kind of value a variable can hold and determine what operations can be performed on it. In Python, common data types include integers, floats, strings, booleans, lists, tuples, and dictionaries. Each type helps store and manipulate data in a specific way.

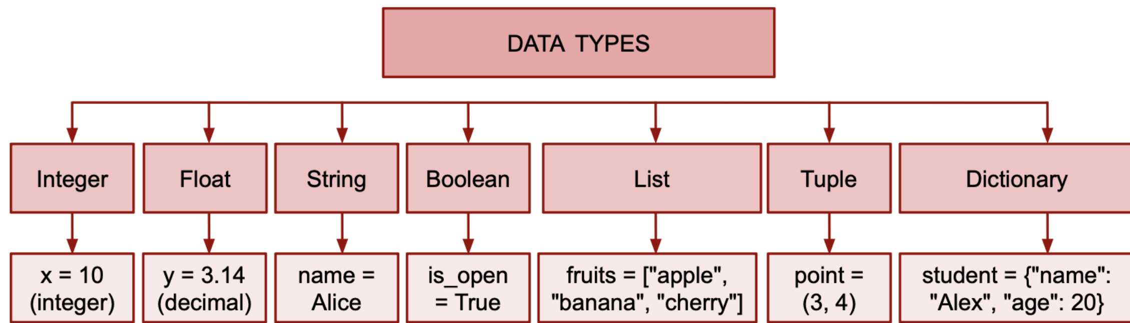

Figure 27: Overview of fundamental data types in Python, including examples of each category: Integer, Float, String, Boolean, List, Tuple and Dictionary. These data types represent the basic building blocks for storing and manipulating information in Python.

Concatenation refers to the process of joining multiple sequence data into a single one, such as joining two strings. Python does not allow implicit conversion between strings and other types for concatenation, so you need to do data conversion before concatenating.

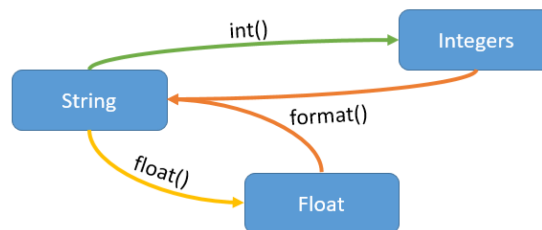

Figure 28: Type conversion between strings, floats, and integers in Python [**gis-precision**]

```
# Author: Ilan
# Date: 04/08/2024
'''The program is intended to output a message with a score'''

marks = 70
print("You scored " + marks)
```

Figure 29: Code snippet with a string-integer concatenation error

Explanation: In the above program, the attempt to concatenate a string (`"You scored "`) with an integer (e.g., `70`) directly results in a `TypeError`.

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[4], line 6
      3 '''The program is intended to output a message with a score'''
      5 marks = 70
----> 6 print("You scored " + marks)

TypeError: can only concatenate str (not "int") to str
```

Figure 30: Python traceback showing a `TypeError` due to incompatible concatenation on line 6

Solution: To resolve this, convert the non-string operand to a string (in a process known as casting) using the `str()` function, as shown here:

```
# Author: Ilan
# Date: 04/08/2024
'''The program is intended to output a message with a score'''

marks = 70
print("You scored " + str(marks))
```

```
You scored 70
```

Figure 31: Code snippet of the corrected code (with casting before concatenation) and output

Alternative Solution: You can use formatted string literals (f-strings) to make your code more readable, as they automatically handle any variable conversion.

```
# Author: Ilan
# Date: 04/08/2024
'''The program is intended to output a message with a score'''

marks = 70
print(f"You scored {marks}")
```

```
You scored 70
```

Figure 32: Code snippet of the corrected code (using f-strings), along with the produced output

5.6 Incompatible Casting

As previously noted, casting involves converting a variable from one data type to another. For example, the integer value 17 can be converted into a string "17" using the `str(17)` function, into a float 17.0 using `float(17)`, and even into a Boolean value `True` using the `bool(17)` function. In the following example, we attempt to convert a non-numeric string to an integer.

```python
# Author: Ilan
# Date: 17/08/2024
'''The program attempts to convert a string containing
non-numeric characters to an integer.'''

todays_wordle = "STORM"
int(todays_wordle)
```

Figure 33: Code snippet illustrating an attempt to convert a non-numeric string to an integer

Explanation: Here, attempting to cast a non-numeric string into an integer throws a `ValueError` (i.e., the function received an argument of the correct type but with an 'inappropriate' value).

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[2], line 7
      3 '''The program attempts to convert a string containing
      4 non-numeric characters to an integer.'''
      6 todays_wordle = "STORM"
----> 7 int(todays_wordle)

ValueError: invalid literal for int() with base 10: 'STORM'
```

Figure 34: Python traceback showing a `ValueError` due to incompatible casting on line 7

This issue frequently occurs implicitly when users are prompted for input and casting is performed on that input in the same line. For example:

```python
# Author: Ilan
# Date: 17/08/2024
'''The program collects a postcode as a string and converts
the duration of residence (in months) to an integer.'''

postcode = input("What is your postcode? ")
duration = int(input("How many months have you lived here? "))
```

```
What is your postcode?  E1 4NS
How many months have you lived here?  twenty-four
```

Figure 35: Code snippet of program that performs casting immediately following user input

Explanation: In this example, a `ValueError` is raised since the entered `duration` is a string literal that cannot be turned into an integer. While we understand that "twenty-four" represents `24`, Python treats "twenty-four" as a meaningless string, similar to entering "Queen Mary" in response to a question about the duration of residence. The string cannot be converted into an integer.

Tip: *When converting data, always ask: "What type of value am I expecting here?"; planning variable types early helps prevent many runtime errors.*

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[7], line 7
      3 '''The program collects a postcode as a string and converts
      4 the duration of residence (in months) to an integer.'''
      6 postcode = input("What is your postcode? ")
----> 7 duration = int(input("How many months have you lived here? "))

ValueError: invalid literal for int() with base 10: 'twenty-four'
```

Figure 36: Python traceback showing a `ValueError` due to casting a non-numeric string on line 7

Solution: To avoid this error, you can use two approaches:

1. Provide a clearer prompt that specifies an integer is required (e.g., "Please enter a number, such as 24, not text like 'twenty-four'").
2. Use a `try-except` block to handle exceptions and prompt the user to enter a valid integer if the input is not a number. When accompanied with a `while`-loop, this ensures that the user is repeatedly prompted until a valid integer is provided.

Question: In the above example, would entering `28` crash the program? What about `20` or `"  12"`?

5.7 Indexing Errors

Consider a simple data structure like an array. Individual elements within this array can be accessed using their index number. Remember, Python uses zero-based indexing, so the index number of the first element is 0, not 1. If you want to access the 5th element in an array named `students`, you would use `students[4]`. The array, `students`, is populated with 10 elements, where index numbers range from 0 to 9. If we mistakenly ignore zero-based indexing and use `students[5]` to access the 5th element (when we actually meant the 6th element), the code will run but will not produce the intended results. This is known as a logical error, which is often harder to detect than syntax or runtime errors. Consider the short program:

```
# Author: Ilan
# Date: 19/08/2024
'''The program defines a list of student names and attempts
to print the 10th student in the list.'''

students = ["co","ha","sa","is","ya","kr","is","hn","il","an"]
print("The 10th student in list: ", students[10])
```

Figure 37: Code snippet attempting to access an out-of-bounds index in a list

Explanation: Attempting to access the 10th element using `students[10]`, this will result in a runtime `IndexError` because index 10 is out of the list's range.

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[9], line 7
      3 '''The program defines a list of student names and attempts
      4 to print the 10th student in the list.'''
      6 students = ["co","ha","sa","is","ya","kr","is","hn","il","an"]
----> 7 print("The 10th student in list: ", students[10])

IndexError: list index out of range
```

Figure 38: Python traceback showing an `IndexError` due to the use of index 10 on line 7

Solution: This is an easy fix. Remember, the 10th element has an index of 9, so use:
`print("The 10th student in list: ", students[9])` to access it correctly.
Note: Indexing with floating-point numbers (e.g., 5.5), would also crash the program but by generating a `TypeError` (rather than an `IndexError` as in the previous case) because indices for arrays must be of integer type only!

Tip: *When working with lists, you can use the* `len()` *function to check how many elements it contains. This helps you avoid using an index that is out of range. For example:* `print(len(students))`.

6 JupyterHub specific issues

Not running all code cells after starting/restarting the kernel.

This mistake usually leads to variables not being updated/defined, causing errors when the code is run.

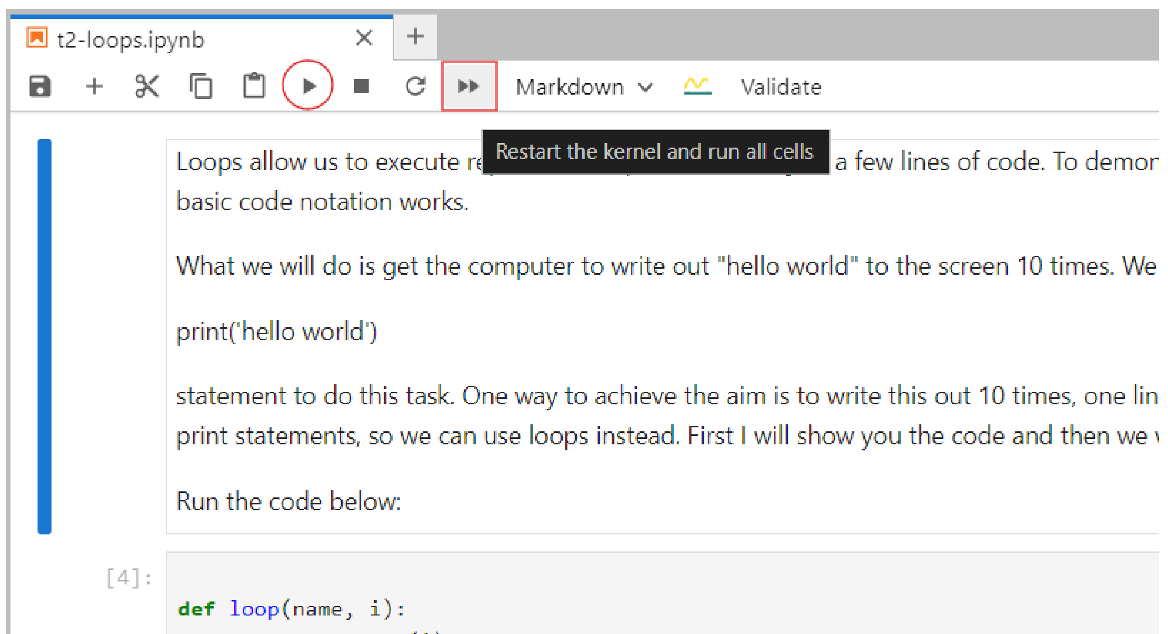

Figure 39: *Circle* - Run current cell, *Square* - Restart kernel and run all cells

A way to mitigate this is to use the button shown in the red box (39), this restarts the kernel as well as re-running all cells in the process. However, if you do not want to run all cells, then the button in the red circle can be used to run selected cells individually.

7 Practice: Find the error and correct it!

Q1)Identify the **single** error in the following code snippet.

```
# Author: Muhie
# Date: 19-08-2024
# Find the error: Q1

if 4 > 3:
    print("I don't like cheese.")
  print("I like cheese.")
```

Q2) Identify **three** errors in the following code snippet.

```
#Author: Yash Vaghela
#Date: 19/08/2024
# Question 2
# Program should print Hello World! 3 times

greeting = "Hello World!"
def greet(input):
    for i in range(3)
    print(input)
    i + 1

greet(greeting)
```

Q3) Identify **four** errors in the following code snippet.

```python
#Author: Yash Vaghela
#Date: 20/08/2024
# Question 3
# Program should check the student score, and output a corresponding grade as a print statement.

student_score = random.randint(0, 100)
import random

def student_grade(score):
    if score >= 90:
        grade = "A"
    elif score >= 80:
        grade = "B"
    elif score >= 70:
        grade = "C"
    elif score >= 60:
        grade = "D"
    elif score < 60:
        grade = "F"
    return grae

print("Student's score is " + student_score + " and grade is " + student_grade(student_score))
```

Q4) Identify **five** errors in the following code snippet.

```python
# Author: Ilan
# Date: 03/09/2024
''' Find the Error: Q4
The program performs a linear search by iterating through
each element in the list to determine if the desired item
is present. It then displays an appropriate message based
on whether the item is found or not.'''

found = False
numbers_list = [28, 30, 22, 4, 17]
search_for = 16

for number in numbers_list:
    if numbre == search_for:
    found = True

if found = True:
    print(search_for, "is in the list of numbers.")
else
    print(search_for + " is NOT in the list of numbers.")
```

Q5) Identify **four** errors in the following code snippet.

```python
# Author: Ilan
# Date: 03/09/2024
''' Find the Error: Q5
The program calculates the area of a circle using a function
that takes a given radius as input and outputs the result.'''

def calculate_area(radius)
    pi = 3.14159
    area = pi * radius ^ 2
    return area

radius = "5"
area = calculate_area(radius)
print("The area of the circle is: " + area)
```

Q6) Identify **five** errors in the following code snippet.

```python
# Author: Ilan
# Date: 03/09/2024
''' Find the Error: Q6
The game prompts the user to guess a number between
1 and 100, giving feedback on whether the guess is too high or
too low until the correct number is found.'''

import math
number_to_guess = random.randint(1, 100)
guess = 30

while guess == number_to_guess:
    guess = int(input("Guess a number between 1 and 100: "))
    if guess < number_to_guess:
        print("Too high! Try again.")
    elif guess > number_to_guess:
        print("Too low! Try again.")
    else:
        print("Congratulations! You guessed the number.")
```

8 Answers to error practice questions

Q1) **Explanation:** The indentation (syntax) error can be solved in two ways:

1. By adding two more white-spaces to contain the `print("I like cheese.")` line within the `if statement`.

2. By removing the two white-spaces to contain the `print("I like cheese.")` line outside of the bounds of the `if statement`.

Solution:

```
# Author: Muhie
# Date: 19-08-2024
# Find the error: Q1 Answer

if 4 > 3:
    print("I don't like cheese.")
    print("I like cheese.")
    """
    You must stay
    consistent with indentation
    python defaults are set to exactly one
    tab key or 4 spaces """
```

```
I don't like cheese.
I like cheese.
```

Q2) **Explanation:**

1. The first mistake is that the `for` statement is missing a colon (:) after its declaration. It would be corrected by placing a ":" after the `for` statement, to indicate that the following lines of code belong to it.
2. The second mistake is the missing indent on the `print` statement, therefore it does not recognise that part of the code to belong to the `for` loop. The correction is to indent the `print` statement, to show this block of code belongs to the `for` loop.
3. The third, similarly to the second, requires indentation, as the "`i + 1`" line is required within the `for` loop block of code to further the loop progress.

Solution:

```
#Author: Yash Vaghela
#Date: 19/08/2024
# Question 2
# Program should print 'Hello World!' 3 times.

greeting = "Hello World!"
def greet(input):
    for i in range(3):    # 1) Added colon to the end of the line
        print(input)    # 2) Indented line
        i + 1    # 3) Indented line

greet(greeting)
```

```
Hello World!
Hello World!
Hello World!
```

Q3) **Explanation:**

1. The first mistake is the incorrect order of importing functions, as this leads to the functions not being recognised and triggering a `NameError`. The correction would be to move the `import` statement to the top of the code, so that it is the first portion of the code to run. Therefore the functions are properly imported.

2. The second mistake is unnecessary code through using one too many `elif` statements. So to simplify the grade calculator function the last `elif` statement can be replaced with an `else` statement.

3. The next mistake is a spelling mistake within the `return` statement, causing a `NameError`. The correction would be to rectify the spelling, allowing the code to run without an error occurring.

4. The final mistake is a concatenation error in the `print` statement, as it is not possible to concatenate strings and integers using "+". The correction would be to combine them using commas (,), or to use an *f-string*.

Solution:

```python
#Author: Yash Vaghela
#Date: 20/08/2024
# Question 3
# Program should print 'Hello World!' 3 times.

import random    # 1) Position of the import line corrected
student_score = random.randint(0, 100)

def student_grade(score):
    if score >= 90:
        grade = "A"
    elif score >= 80:
        grade = "B"
    elif score >= 70:
        grade = "C"
    elif score >= 60:
        grade = "D"
    else:    # 2) Simplified the function
        grade = "F"
    return grade    # 3) Spelling mistake corrected from 'grae' to 'grade'

print("Student's score is ", student_score," and grade is ", student_grade(student_score))
# or, print(f"Student's score is {student_score} and grade is {student_grade(student_score)}")

# 4) Corrected the concatenation error using commas or an f-string
```

```
Student's score is  81  and grade is  B
```

Q4) **Explanation:**

1. The variable `number` was misspelled in the condition statement within the first `if` statement. As a result, Python raises a `NameError`, assuming the variable was not declared.

2. The second error occurs on the following line, where the indentation is incorrect. It's worth noting that if another statement within the `if` block was correctly indented, but the line containing `found = True` remained misaligned, the code would run without raising an `IndentationError`. However, the program's logic would not function as expected.

3. When the content of the `found` variable is compared against `True`, an assignment operator was incorrectly used instead of the equality operator (==). This throws a `SyntaxError`.

4. A missing colon after the `else`.

5. In the last line of the code, an integer (stored in `search_for`) was concatenated with a string. This incompatible concatenation would have resulted in a `TypeError`. Note that this issue does not arise within the `if` block, where a comma was used to merge the two values, so there was no need for type casting.

Solution:

```python
# Author: Ilan
# Date: 03/09/2024
''' Find the Error Answer: Q4'''

found = False
numbers_list = [28, 30, 22, 4, 17]
search_for = 16

for number in numbers_list:
    if number == search_for: # 1) 'number' was spelled wrong
        found = True # 2) this line of code was not indented properly

if found == True: # 3) an assignment operator was used previously
    print(search_for, "is in the list of numbers.")
else: # 4) missing colon after the 'else'
    # 5) string-integer concatenation
    print(str(search_for) + " is NOT in the list of numbers.")
```

```
16 is NOT in the list of numbers.
```

Q5) **Explanation:**

1. A missing colon after the `)` in the function definition.
2. The `^` symbol was used in place of the `**` operator for exponentiation.
3. When the function was called, a string `radius` value was passed. Since strings cannot be directly used for mathematical operations, they must be converted to integers. In the solution below, we have chosen to perform this conversion within the function call, but it could also have been done later, such as on the line where the `area`) is calculated.
4. The value returned for the `area` is of type float. It must be converted to a string before concatenating it with the other string value in the final print statement. Otherwise, this line would result in a `TypeError`.

Solution:

```python
# Author: Ilan
# Date: 03/09/2024
''' Find the Error Answer: Q5'''

def calculate_area(radius): # 1) missing colon after function definition
    pi = 3.14159
    area = pi * radius ** 2 # 2) ^ was used as exponentiation operator
    return area

radius = "5"
area = calculate_area(int(radius)) # 3) a string radius was passed initially
# 4) since area is a float, it must be cast to a string before concatenation
print("The area of the circle is: " + str(area))
```

```
The area of the circle is: 78.53975
```

Q6) **Explanation:**

1. The `math` library was incorrectly imported. The correctly library should be `random` since its needed to generate a random number.
2. The second error is that the default `guess` was incorrectly initialised to 30. This breaks the program's logic, because if all other errors were fixed and the random number generated happened to be 30, the condition in the `while` loop (`guess != number_to_guess`) would immediately evaluate to `False`. As a result, the loop wouldn't run, and the game would end before the user even gets a chance to make any guesses, incorrectly assuming the number has already been guessed. To fix this, assign a value to `guess` that is outside the valid range (e.g., 0-100). A value like -1 would be a reasonable choice.
3. The third error is that the condition within the `while` loop is set up incorrectly as `guess == number_to_guess`. This means the loop only runs when the guess is equal to the number to guess, which is the opposite of the intended behaviour. To correct this, revise the condition to `guess != number_to_guess`, ensuring that the loop continues to execute until the guess matches the target number.
4. The incorrect use of < instead of > in the `if` statement results in incorrect feedback. Specifically, it directs the user to guess a number lower than their current guess, when they actually need to guess a value that is higher.
5. Just like the previous error, using > instead of < leads to incorrect feedback being given to the user.

Solution:

```python
# Author: Ilan
# Date: 03/09/2024
''' Find the Error Answer: Q6'''

import random # 1) incorrect library (math) was imported
number_to_guess = random.randint(0, 100)
guess = -1 # 2) default guess was set to 30 (in range)

while guess != number_to_guess: # 3) condition was set-up incorrectly
    guess = int(input("Guess a number between 1 and 100: "))
    if guess > number_to_guess: # 4) incorrect operator (< was used)
        print("Too high! Try again.")
    elif guess < number_to_guess:  # 5) incorrect operator (> was used)
        print("Too low! Try again.")
    else:
        print("Congratulations! You guessed the number.")
```

```
Guess a number between 1 and 100:  70
Too low! Try again.
Guess a number between 1 and 100:  94
Congratulations! You guessed the number.
```

9 Dry run template

Below is an example dry run table of a program, with the first iteration completed. Can you finish it? *Hint: You need to add more rows*:

```python
x = 0
while x < 3:
    if x % 2 == 0:
        print(x, "is divisible by 2 with no remainder")
    x = x + 1
```

Figure 40: Dry run this program, answers can be found on QM+

Line	x	is $x < 3$?	$x \% 2 == 0$?	Output
1	0			
2		true		
3			true	
4				0 is divisible by 2 with no remainder
5	1			
2				

10 I NEED MORE HELP, WHAT DO I DO?

10.1 Bonus: help() built in python function (method)

Sometimes if you are slightly unsure of what a function in Python might do, you can use the `help` method to gain a quick picture of what it does by using the help method you can do this by calling `help(your method you want help with)`.

Example:

```
# Muhie Al Haimus
# 4/8/24
# A Python script to test out the built-in help method
help(round)
```

Output:

```
Help on built-in function round in module builtins:

round(number, ndigits=None)
Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.  Otherwise
the return value has the same type as the number.  ndigits may be negative.
```

10.2 Written Resources

Python reference guide. [8] - A great way to learn syntax.
Stack overflow. [9] - A web forum that has answers to almost every problem in Python you may encounter.

10.3 Video resources

Tech With Tim [10] - Introductory Python tutorials., Numpy tutorials.

10.4 What's next?

Learn Markdown: this is essential to learn to create readable code documents with Juypter notebooks. It is also really useful for note-taking as it has $LaTeX$ integration for mathematical equations a guide to markdown can be found here. [11]

Learn Latex: Allows you to produce mathematical equations and documents like the one you are reading now! If you are looking to do a Masters or PhD, you should start learning it earlier rather than later so that you can effortlessly produce good-looking professional documents you can do this online using Overleaf. [12]

Learn OOP: Object-oriented programming (OOP) is an advanced style of programming. It is the standard for programming in industry for: GUI's, Games and Robotics. So far, you would have only come across procedural programming which is a really good foundation for understanding the fundamentals of a programming language. However, it is not really suitable for large code-bases that need to be maintained by large teams as it creates large amounts of code duplication.

11 Dry Run Table Answers

```
x = 0
while x < 3:
    if x % 2 == 0:
        print(x, "is divisible by 2 with no remainder")
    x = x + 1
```

Figure 41: Dry run this program, answers can be found on QM+

Line	x	is $x < 3$?	$x \% 2 == 0$?	Output
1	0			
2		true		
3			true	
4				0 is divisible by 2 with no remainder
5	1			
2		true		
3			false	
5	2			
2		true		
3			true	
4				2 is divisible by 2 with no remainder
5	3			
2		false		

References

[1] Bits and Bytes TVO. *Interpreters and Compilers (Bits and Bytes, Episode 6)*. URL: https://www.youtube.com/watch?v=_C5AHaS1mOA. (accessed: 19.08.2024).

[2] Muhie Al Haimus. *EMS412U: Python install guides*. URL: https://www.youtube.com/playlist?list=PLVTKec-v1Xhsq0B_q3NbLEAafPlh_3XhE. (accessed: 5.09.2024).

[3] Python Software Foundation. *Download Python — Python.org*. URL: https://www.python.org/downloads/. (accessed: 21.08.2024).

[4] Visual Studio Code. *Download Visual Studio Code - Mac, Linux, Windows*. URL: https://code.visualstudio.com/download. (accessed: 21.08.2024).

[5] Jetbrains s.r.o. *Pycharm: The Python IDE for data science and web development*. URL: https://www.jetbrains.com/pycharm/. (accessed: 21.08.2024).

[6] Anaconda Inc. *Download Now — Anaconda*. URL: https://www.anaconda.com/download/success. (accessed: 21.08.2024).

[7] Jamie Wagner. *Asking Siri To Divide 0 x 0*. URL: https://www.youtube.com/watch?v=QVchft8FoYQ. (accessed: 21.08.2024).

[8] W3Schools. *Python Syntax*. URL: https://www.w3schools.com/python/python_syntax.asp. (accessed: 21.08.2024).

[9] Stack Overflow. *Stack Overflow*. URL: https://stackoverflow.com/. (accessed: 20.08.2024).

[10] Tech With Tim. *Python Courses Tutorial*. URL: https://www.youtube.com/@TechWithTim. (accessed: 21.08.2024).

[11] John Gruber. *Markdown Cheat Sheet*. URL: https://www.markdownguide.org/cheat-sheet/. (accessed: 21.08.2024).

[12] Overleaf. *Overleaf, Online LaTex Editor*. URL: https://www.overleaf.com. (accessed: 21.08.2024).