Preparation for Assessment

Objective:

The purpose of this document is to address common Python Assessment pitfalls, guide the reader through some of the most challenging Python concepts, and support consultants with their assessment preparation.

1.Python Assessment Pitfalls:

Please read the below points and ensure comprehensive understanding of what NOT to do on the assessment to avoid unneccessary mark loss!

1. Do **NOT** change the name of your submission file - stick to the **submission.ipynb** file name
2. Ensure that **ALL SUBMIT CELLS** run **WITHOUT CAUSING ERRORS** - a single error in your submit cell can prevent your trainer from generating your assessment feedback and thus cost you marks
3. Ensure to write your solutions in the **SUBMIT CELLS** - these are the only cells which your trainer will be assessing
4. Do **NOT** create more than 1 **SUBMIT CELL** per question
5. Do **NOT** delete the comments, identifying the cells as **SUBMIT CELLS**
6. Ensure to **ONLY** write your **function definition** or **class definition** in the **SUBMIT CELLS**:
    1. They should **NOT** contain any global variables
    2. They should **NOT** contain any function calls or instances of classes
    3. Good example:

```
# submit (DO NOT REMOVE THESE COMMENTS. REPLACE THE PASS STATEMENT WITH YOUR
SOLUTION)
# ---------------------------------------------------------------------------
-------------------------
def keep_only_strings(x):
    string_list = []
    for item in x:
        if type(item) == str:
            string_list.append(item)
        else:
            pass
    return string_list
```

    4. Bad example:

```
# submit (DO NOT REMOVE THESE COMMENTS. REPLACE THE PASS STATEMENT WITH YOUR
SOLUTION)
# ---------------------------------------------------------------------------
-------------------------
def keep_only_strings(x):
    string_list = []
    for item in x:
        if type(item) == str:
            string_list.append(item)
        else:
            pass
    return string_list

example_list = [1,2,3,'hello', 'world'] # This is a global variable -
prohibited to use here
keep_only_string(example_list) # This is a function call - prohibited to use
here
```

7. Ensure to use the **required function/ Class names** - a single typo can cost you 5 marks
8. Ensure to **read the questions CAREFULLY** - in particular, pay extra attention to:
    1. the **number of function parameters**
    2. the **type of objects to be returned**
    3. the **order of returned objects** if the question asks you to define a function, returning multiple objects
9. When a function is required to **return an object**, do **NOT** use `print()` - the `print()` function does NOT return a tangible object at the place of its call, rather ONLY displays the value of the print argument:
    1. Good example:

```
# submit (DO NOT REMOVE THESE COMMENTS. REPLACE THE PASS STATEMENT WITH YOUR
SOLUTION)
def keep_only_strings(x):
    string_list = []
    for item in x:
        if type(item) == str:
            string_list.append(item)
        else:
            pass
    return string_list
```

    2. Bad example 1:

```
# submit (DO NOT REMOVE THESE COMMENTS. REPLACE THE PASS STATEMENT WITH YOUR
SOLUTION)
```

```
def keep_only_strings(x):
    string_list = []
    for item in x:
        if type(item) == str:
            string_list.append(item)
        else:
            pass
    print(string_list) # This is wrong - print() only displays the value
```

3. Bad example 2:

```
# submit (DO NOT REMOVE THESE COMMENTS. REPLACE THE PASS STATEMENT WITH YOUR
SOLUTION)
def keep_only_strings(x):
    string_list = []
    for item in x:
        if type(item) == str:
            string_list.append(item)
        else:
            pass
    return print(string_list) # This is wrong - this will return None
```

10. When a question asks you to **return True, False or None** , do **NOT** enclose these in single quotes, do **NOT** misspell the values and **DO** ensure you use the correct capitalisation (i.e. **uppercase first letter**):

1. Good example 1:

```
def f(x):
    if x>5:
        return True
    else:
        return False
```

2. Good example 2:

```
def f(x):
    if x>5:
        return x
    else:
        return None
```

2. Bad example 1:

```
def f(x):
    if x>5:
```

```
        return 'True' # This is wrong - this will return a string
    else:
        return 'False' # This is wrong - this will return a string
```

2. Bad example 2:

```
def f(x):
    if x>5:
        return Treu # This will raise an error - spelt incorrectlySPELT
INCORRECTLY
    else:
        return  false # This will raise an error - Python is case sensitive
```

11. When defining a function, do **NOT** write a `return` statement before any other important statements in your function body. The `return` statement puts **an END** to your function call - everything underneath it will **NOT BE EXECUTED**.

12. When defining a function, do **NOT** write a `return` statement **inside a `for` loop**. When Python encounters the `return` statement in the first iteration of the `for` loop, it will **exit** the function call, and so your for loop will only iterate once (defeating the purpose of looping).

    1. Good example:

```
def keep_only_strings(x):
    string_list = []
    for item in x:
        if type(item) == str:
            string_list.append(item)
        else:
            pass
    return string_list
```

    2. Bad example:

```
def keep_only_strings(x):
    string_list = []
    for item in x:
        if type(item) == str:
            string_list.append(item)
        else:
            pass
        return string_list # This is wrong - Python will execute only 1
iteration
```

13. When a question states that a function **accepts an input argument**, this does **NOT** mean that the function should be using the `input()` function! An input is the other name for argument! If the question really requires the use of the `input()` function, it will read ***write a function which asks the user for their input***

2. I am struggling to translate the Question into Python code - HELP!

Although Python is a programming language, in which every command is an English word, alluding to its purpose and function, mastering the skill of translating requirements into code requires time and practice, and can be overwhelming at first!

Below are several battle-tested strategies, which you can apply immediately and improve your programming skills!

2.1. Example 1 - Build a Methodology:

Question Set-Up:

Write a function called `search_address_book` that accepts two arguments - an `address_book` dictionary in which the keys are the names and the values are the numbers (stored as strings), and a `search_string`. The function should **return a dictionary** that contains all the entries in the original address book, for which the **name starts with the `search_string`**.

- **Step 1** - dissect the question into separate logical steps (parts):
  - *Write a function called `search_address_book`* - translates into `def search_address_book():`
  - that accepts two arguments - an `address_book` and a `search_string`* - translates into `def search_address_book(address_book,search_string):`
  - *The function should return a dictionary* - this translates into 2 things:
    - creating a new empty dictionary right at the beginning of the function body - `new_dict = {}`
    - having a `return` statement at the end of the function body - `return new_dict`
  - *a dictionary that contains all the entries in the original address book, for which the name starts with the `search_string`* - this is where the bulk of our logic/code will be:
    - first we will have to assess each name in the address book individually - this translates into a `for` loop
    - since the names in the address book are the keys, we have to iterate through the dictionary keys - this translates into `for name in address_book:` (or `for name in addrress_book.keys()` or `for name, number in address_book.items()`)
    - next, we have to check if the name begins with the `search_string` - this translates into using an `if` statement, in combination with `.find()` string method
    - if the name begins with the string, I would like to append the name and its number to the new dictionary - this translates into:

```
if name.find(search_string) == 0:
    new_dict[name] = number
```

    - if the name does NOT begin with the given string, I would like to NOT add it to the new dictionary (i.e. take no action) - this translates into:

```
if name.find(search_string) == 0:
    new_dict[name] = number
else:
    pass
```

- **Step 2** - combine all of the above steps in a single solution:

```
def search_address_book(address_book, search_string):
    new_dict = {}
    for name, number in address_book.items():
        if name.find(search_string) == 0:
            new_dict[name] = number
        else:
            pass
    return new_dict
```

- **Step 3** - come up with an example values for `address_book` and `search_string` to test how your function performs:

```
example_book = {'Anna Scott': '07599 247658', 'Dan Brown': '0888 722 795', 'Anna Mcdonald': '01234456'}
search_address_book(example_book, 'Anna')
```

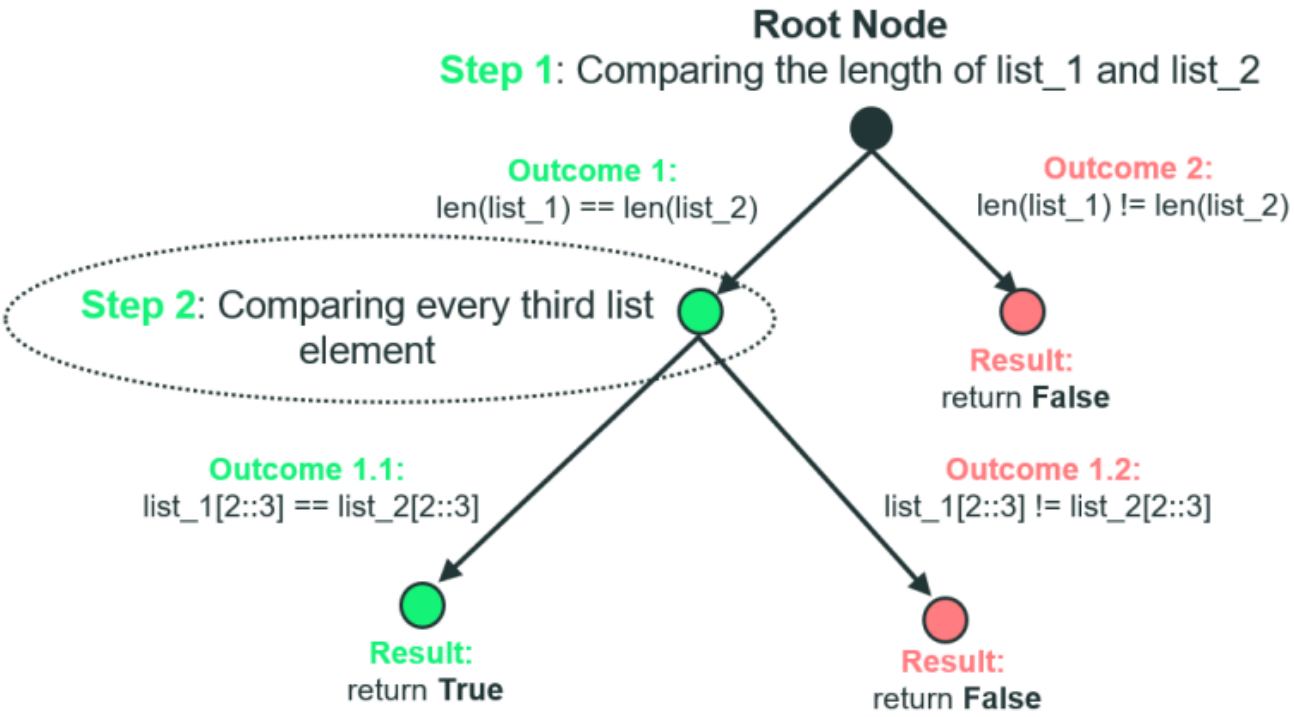
2.2 Example 2 - Try to Visualise:

Question Set-Up:

Write a function called `every_third_equal` that takes in two lists and returns `True` if the lists are of the same length and every third element in these lists is equal to each other, and otherwise, returns `False`.

- **Step 1** - Identify the main decisions to be made in the function:
    - First we have to assess the length of the two lists and check if they are equal.
    - Second, we have to assess if the 3rd, 6th, 9th, etc, element in the first list is equal to the 3rd, 6th, 9th, etc, element in the second list (i.e. that every third element is the same).

- **Step 2** - Visualise a decision tree:



- **Step 3** - Translate into Code:

```
def every_third_equal(list_1, list_2):
    if len(list_1) != len(list_2):
        return False
    else:
        if list_1[2::3] == list_2[2::3]:
            return True
        else:
            return False
```

2.3 Main Tips to Follow:

- If you have to account for different **conditional** situations, use an if statement, or an if...else statement, or an if...elif...else statement.
- If fulfillment of one **conditional** statement leads you to assessing a **second conditional** statement, you have to **nest** an if..else statement inside another if...else statement.
- If you have to **iterate** through **an object**, use a for loop.
- If you have to **iterate** through **a dictionary**, assess which part of the dictionary you want to work with on each iteration:
    - If you have to work with the keys only, use `for key in my_dict` or `for key in my_dict.keys()`.
    - If you have to work with the keys and values, use `for key, value in my_dict.items()`.
- If you have to create and return an object as part of your function, create an **empty container** of **the required data type** and populate it selectively. For example, create an empty list (`empty_list = []`) and append to it only the elements which fulfill a criterion with `.append()`.

- If you have to **handle exceptions**, use a try...except statement.
- If you have to **raise an exception**, use a raise statement.
- If you have to assess the data type of an object, use `type()` built-in function. For example:

```
x = 10
type(x) == int # This will return True

x = 'hello guys'
type(x) == str # This will return True

x = 7.5
type(x) == float # This will return True

x = True
type(x) == bool # This will return True

x = [1, 2, 3]
type(x) == list # This will return True

x = {'key1': 1, 'key2': 2}
type(x) == dict # This will return True

x = (1, 2, 3)
type(x) == tuple # This will return True

x = {1, 2, 3}
type(x) == set
```

3. I cannot make sense of Classes - HELP!

Classes can be an overwhelming topic - it takes us away from our comfort zone and requires us to act as designers, shaping a new data type from scratch. But just like anything else in Python, if you break down Classes into small steps, building them is just as easy as building any other code!

Recall that defining a Class is the same as defining a new data type. We have been working mostly with pre-built data types - integers, strings, lists, dictionaries, etc., so we never had to bother with defining how objects of these data types should behave.

But what if we want to create our own data type? A data type that no one but us has access to? The steps are easy and once followed in the right order, no Class definition looks challenging anymore!

3.1 Classes - Terms to Know:

- **Class:** a data type; a blueprint (or design), shaping the form and behavior of objects, belonging to this data type.
- **Instance:** a representative object of the Class; a tangible object that we can construct and work with.
- **Instance Attribute:** a value, or a piece of information, bound to a Class instance; examples - age, color, memory space, etc.

- **Instance Method:** a function, bound to a Class instance; example - to greet, to log a file, to compare two values and return the highest, etc.

3.2 Defining a Class - Steps:

- **Step 1** - Initiation of the Class definition - `class ClassName:`
  - begins with the keyword `class`
  - follows with the class name, written in **UpperCamelCase** - example `MyLogger`, `Consultant`, `Teacher`
  - ends with a colon :
- **Step 2** Define the **init** method (also known as *Constructor* method) - `def __init__(self):`
  - this method gets called and executed everytime you create a new class instance
  - the purpose of this method is to **construct the instance and the instance attributes**
  - it takes a special argument **self**
  - pay attention to the method name - the init is surrounded by **double underscore**
  - methods are defined the same way as any other function! Example:

```
class Dog:
    def __init__(self, age):
        self.age = age
```

- **Step 3** - Define the **instance attributes** - example: `self.age = age`
  - this is done in the **init method**
  - to define an instance attribute, we use the `self.` syntax, followed by the attribute name
  - to **assign** a value to the instance attribute, we use the `=` sign
  - in the example above the attribute, called `age` is assigned to the value, stored in the **init argument** called `age`
  - when defining instance attributes, we can assign static or dynamic values, example:

```
class Dog:
    def __init__(self, age):
        self.age = age
        self.sleeping_place =  'sofa'
```

  - above we defined 2 **attributes** called `age` & `sleeping_place`
  - we assigned a static value `'sofa'` to `sleeping_place` and a dynamic value `age` to the attribute `age`
- **Step 4** - Define the **instance methods** - example: `def is_old(self):`
  - methods are defined the same way as functions
  - only difference is that they accept a special first argument, called **self**
  - methods can perform anything we like - return objects, change values of instance attributes, or display messages, example:

```
class Dog:
    def __init__(self, age):
        self.age = age
        self.sleeping_place =  'sofa'

    def is_old(self):
        if self.age >= 10:
            return True
        else:
            return False

    def change_sleeping_place(self, new_place):
        self.sleeping_place = new_place
```

- above we defined a Class called **Dog**
- the **init** method constructs 2 **instance attributes** - `age` and `sleeping_place`
- the `is_old()` **method** is defined to check if the instance is old and return `True` or `False`
- this method accesses the instance attribute `age` from within its body and compares it to 10
- to **access an instance attribute from within a method** we use the syntax `self.attribute_name`
- the `change_sleeping_place()` **method** is defined to reassign the value of the instance attribute `sleeping_place`
- to **reassign a value of an instance attribute from within a method** we use the syntax `self.attribute_name = new_value`

3.3 Creating Class Instances - Steps:

- **Step 1** - Create an instance to the Class
  - Once the class is defined, we can create instances to it, example:

```
troy = Dog(8)
```

  - above we created a variable `troy`, referencing an object, which is an instance of the Class `Dog`
  - in the brackets after the class name we have to enter **specific values for ALL arguments in the init method except self** - for `troy` this is `8` and this is the value for the argument `age`
  - the `init` method will then take this value and **assign it** to the attribute `age`.
- **Step 2** - Accessing **Instance Attributes** - example: `troy.age`
  - to retrieve information, bound to an instance to a Class, we can access their **attributes**, example:

```
troy.age # This returns 8

troy.sleeping_place # This returns 'sofa'
```

- **Step 3** - Calling **Instance Methods** - example `troy.is_old()` or `troy.change_sleeping_place('bed')`
    - to execute a method onto a given instance, we can **call the method**
    - in the brackets after the method name, we have to enter **specific values for ALL arguments in the method, except self (if the method definition asks for an argument**, example:

```
troy.is_old() # No argument required. This returns False, as troy's age is 8, which is less than 10
troy.change_sleeping_place('bed') # One argument required. This will assign 'bed' to troy's instance attribute `sleeping_place`s age is 8, which is less than 10
```

To check that the `change_sleeping_place()` method worked, we can access troy's attribute `sleeping_place` and observe it's value

```
troy.sleeping_place # This will output 'bed'
```

3.4 Syntax - Tips and Tricks:

When Defining a Class:

- If you define methods, always put a self argument as the first argument in the definition.
- To define attributes, declare them in the `__init__()` method using syntax `self.attribute_name = value`.
- To access an attribute from within the Class definition, use syntax `self.attribute_name`.
- To call a method from within the Class definition, use syntax `self.method_name()`.

When using the Class, i.e., creating instances:

- To create an instance, use the syntax `instance_name = ClassName()` where you populate the () with specific values for the init arguments, if any arguments are required.
- To retrieve a value from an instance attribute, use syntax `instance_name.attribute_name`.
- To call an instance method, use syntax `instance_name.method_name()` where you populate the () with specific values for the given method's arguments, if any arguments are required. The only argument you should not give a value for in the brackets is `self`.