CMP 4266
Software Development UG1

Lecture – 6

A quick revision

List

- List: an object that contains multiple data items.
  - Element: An item in a list
  - Format: `list = [item1, item2, etc.]`
- To access an element in the list, use [i] where i is the index of the element.
  - `list[i]` refers to the $(i-1)^{th}$ element in the list.
- Lists are **mutable**, which means their elements can be changed.
  - `list[i] = new_value` can be used to assign a new value to a list element.

```
colours = ["red", "blue", "orange", "green", "white", "purple"]
print(colours)
print(colours[2])
print(colours[-2])
print(colours[6])
colours[3] = "black"
print(colours)
```

```
['red', 'blue', 'orange', 'green', 'white', 'purple']
orange
white
IndexError: list index out of range
urple']
```

Function, argument, and variable scope

- For the following codes, `my_num` in the different scopes will no affect each other:

```
def fun1(my_num):
    my_num = 10

def fun2(my_list):
    my_list.append(10)
```

```
my_num = 5
print(my_num)
fun1(my_num)
print(my_num)
```

- More precisely, when you pass a variable as an argument into a function, the argument is actually a copy of the original variable.
- Therefore, any changes made inside the function will not affect the variable values outside of the function.

```
if __name__ == "__main__":
    n = 5
    print(n)
    fun1(n)
    print(n)

    l = [1, 2]
    print(l)
    fun2(l)
    print(l)
```

```
5
5
```

```
[1, 2]
[1, 2, 10]
```

Mystery behaviour of list?

- Quick answer:
  - When a list is passed in as argument, it is itself got passed in not a copy.
  - In other words, if you passed in a list to a function and changed its contents within the functions, such changes will be reflected in original list.

```
def amend_int(my_int):
    my_int = 10

def amend_str(my_str):
    my_str = "Hi"

def amend_list(my_list):
    my_list.append("another item")
```

```
test_num = 5
print(test_num)
amend_int(test_num)
print(test_num)

test_str = "Hello"
print(test_str)
amend_str(test_str)
print(test_str)

test_list = ["Old", "List"]
print(test_list)
amend_list(test_list)
print(test_list)
```

```
5
5

Hello
Hello

['Old', 'List']
['Old', 'List', 'another item']
```

Want to know what actually happened?

- Further readings:
  - https://pythonguides.com/python-pass-by-reference-or-value/
  - https://www.geeksforgeeks.org/is-python-call-by-reference-or-call-by-value/
  - https://www.tutorialspoint.com/pass-by-reference-vs-value-in-python
  - https://www.geeksforgeeks.org/pass-by-reference-vs-value-in-python/

  - Test your understanding with the following function:

```
def amend_list2(my_list):
    my_list = ["new list", "item 2"]
```

  - Try it out and does it behave as you would expect?
  - Note. This is beyond the scope of this module.

Errors in codes

Type of Errors

- Syntax Errors:
  - Common syntax errors could be incorrect indentation, missing elements, misspelling etc. Easiest to find, by code inspection or the interpreter reports it at compilation time.
  - IDE (e.g. Spyder) will not allow you to run the code.

- Runtime Errors:
  - May go unnoticed at compilation time and only revealed at runtime.
  - The code will run but will encounter errors while running.

- Logical Error:
  - Usually the hardest to find. Can be identified by observing/watching the program behaviour.
  - Code will run "as normal" but might not behave "as expected".

Syntax Errors

Syntax Errors - Code inspection

- Python interpreter will find these kinds of errors when it tries to compile your program, and exit with an error message without running anything.
- Syntax errors are mistakes in the use of the Python language.
- It's a good idea to give code a thorough visual inspection first, before trying to compile it.
  - Check for layout/indentation
  - Check for punctuation
  - Check for spelling mistakes

Code inspection: checking for consistency of layout

- Python interpreter uses the indentation to mark the beginning and end of blocks.
- Some of the block control key words in Python are *def*, *for*, *if, else, elif, try, except* etc.
- Example

```
1. def test():
2.         for year in [2000,1903,2008,2009]:
3.     if(isleap(year)):
4.             isornot=""
5.                else:
6.     isornot="not "
7.         print(str(year)+" is "+isornot+"a leap year")
```

How does the interpreter know which statements are controlled by which block controls ? It can't form a layout like this.

Code inspection: correcting block structure

```
1. def test():
2.     for year in [2000,1903,2008,2009]:
3.         if(isleap(year)):
4.             isornot=""
5.         else:
6.             isornot="not "
7.         print(str(year)+" is "+isornot+"a leap year")
```

How many times should the print statement execute ?

- 4

Code inspection: correcting block structure

```
1. def test():
2.     for year in [2000,1903,2008,2009]:
3.         if(isleap(year)):
4.             isornot=""
5.         else:
6.             isornot="not "
7.     print(str(year)+" is "+isornot+"a leap year")
```

How many times should the print statement execute ?

- 1

Code inspection: correcting block structure

```
1. def test():
2.     for year in [2000,1903,2008,2009]:
3.         if(isleap(year)):
4.             isornot=""
5.         else:
6.             isornot="not "
7.             print(str(year)+" is "+isornot+"a leap
year")
```

How many times should the print statement execute ?

- 2

Code inspection: checking punctuation

- Brackets and quotes (i.e. () [] {} "" ' ‘) must appear as matching pair.
- Block control statements **def**, **if**, **for**, **while** etc. must end with a : colon.
- A backslash \ or % escape can make the following character into something different.
- List items are separated using commas (,).

```
1. def test( :
2.     for year in [2000.1903'2008,2009 };
3.         if (isleap(year):
4.             isornot='"
5.         else
6.             isornot="not \"
7.         print(str(year)+" is " &isornot+"a leap year")
```

How many punctuation errors can you spot?

Code inspection: checking punctuation

- Have you find them all?

```
1. def test() :
2.     for year in [2000.1903'2008,2009 };
3.         if (isleap(year)):
4.             isornot='"
5.         else:
6.             isornot="not \"
7.         print(str(year)+" is " &isornot+"a leap year‟)
```

Code inspection: checking spelling

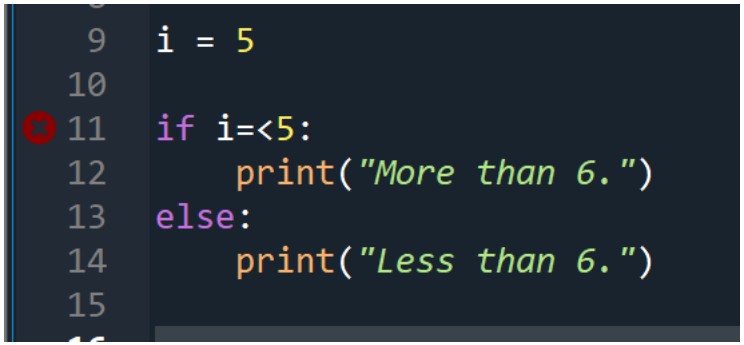
```
1. def test():
2.     for y in [2000,1903,2008,2009]:
3.         if(isleap(year)):
4.             isornot=""
5.         else:
6.             isorno="not "
7.         print(str(yaer)+" is "+issornot+"a leap year")
8. Test()
```

- The defined function name needs to be checked against all calls to it.
- The variables used for year and isornot also need consistent spelling.

Fixing compilation errors 1

- The IDE (e.g. Spyder) will report a location within code where it detects an error preventing compilation.

- The error might be at the place in the code where the error is detected.



Fixing compilation errors 2

- Or the real error might be ***before*** the place in the code where the compiler ***detects*** the error. Can you spot the syntax error in the line above the else?

- We still need to do a careful visual code inspection.

```
 9  i = 5
10
11  if i >= 5:
12      print("More than 6."
13  else:
14      print("Less than 6.")
15
```

Caution!

Syntax error **should never happen** in your submission to the coursework.

Exercise 6.1 – Identifying syntax errors

- Type the following code in Spyder.

```
import calendar

def test( :
    for year in [2000.1903'2008,2009 };
        if (calendar.isleap(year):
            isornot='"
        else
            isornot="not \"
        print(str(year)+" is " &isornot+"a leap year")
```

- Can you fix all the errors with help of the IDE?
- After completion, refer to the previous slides for the answer.

Runtime Error

Runtime errors

- If a program is syntactically correct – that is, free of syntax errors – it will be run by the Python interpreter.

- However, the program may exit unexpectedly during execution if it encounters a *runtime error.*

- Some examples of Python runtime errors:
  - division by zero
  - performing an operation on incompatible types
  - using a variable which has not been defined
  - accessing a list element or object attribute which doesn't exist
  - trying to access a file which doesn't exist

Reading runtime exception tracebacks

- The traceback which occurs when an exception is raised at runtime helps us locate and identify the error causing a crash, but we'll also need to inspect and understand the code which raised it.

```
15
16  my_list = ["week", "5"]
17  print(my_list[0])
18  print(my_list[1])
19  print(my_list[2])
```

```
week
5
Traceback (most recent call last):

  File "D:\Work\OneDrive - Birmingham City University\Teaching\CMP4266 - Computer
Programming\CMP4266\Lab 5\demo.py", line 19, in <module>
    print(my_list[2])

IndexError: list index out of range
```

- Should be prevented by careful coding.

Reading runtime exception

- Sometime, you might receive an runtime error out of you control:

```
 9  i = int(input("Enter a number: "))
10
11  if i >= 5:
12      print("More than 6.")
13  else:
14      print("Less than 6.")
```

```
Enter a number: ten
Traceback (most recent call last):

  File "D:\Work\OneDrive - Birmingham City University\Teaching\CMP4266 - Computer
Programming\CMP4266\Lab 5\demo.py", line 9, in <module>
    i = int(input("Enter a number: "))

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

- Can be handled via exception handling – you will learn it in week 9.

Exercise 6.2 – Identifying syntax & runtime errors

- Identify and fix 10 syntax and runtime errors in the code:
  - ex_6.2.py
- You should put a #comment next to each identified error indicating the error type e.g. #runtime or #syntax
- Once the software is fixed, the software output should look like the following:

```
>>>
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4):3
Enter first number: 5
Enter second number: 7
5 * 7 = 35
>>>
```

- Make sure you are able to do it independently.

Logical Error

Logical Error

- Logical errors are the most difficult to fix.
- They occur when the program runs without crashing, but produces an incorrect result.
- The error is caused by a mistake in the program's logic. Hence, you won't get an error message, because no syntax or runtime error has occurred.
- You will have to find the problem on your own by reviewing all the relevant parts of your code.

- Here are some examples of mistakes which lead to logical errors:
  - using the wrong variable name
  - indenting a block to the wrong level
  - Wrong mathematical calculation
  - making a mistake in a boolean expression

How to find logical error?

- When the implementation is simple, you can do a inspection.

```
 9  i = int(input("Enter a number: "))
10
11  if i > 5:
12      print("More than 6.")
13  else:
14      print("Less than 6.")
```

- But the common approach is to conduct a testing:

```
Enter a number: 7
More than 6.
```

```
Enter a number: 6
More than 6.
```

```
Enter a number: 5
Less than 6.
```

Test Case

- Test Case is the basic component to perform software testing. In the most general form test cases
  - Specifies the actual input values and anticipate output values.
  - Identifies any constrains on the execution of the test case (e.G. Processor speed, available memory).

- Test case execution: is the process of executing the software system
  - Under the constraints specified in the test case.
  - Using the inputs specified in the test case.
  - Observing the results and comparing them to those specified by the test case.
  - If the observed result varies from the anticipated result, then a failure has been detected.

Test Case Design

- Testing cannot guarantee the absence of all defects, so test-case design is important to try to make tests as complete as possible.

  - Write a program to calculate the square root of B * B + 4 *A * C and print out the result.
  - Create a script named **`lab1_expression.py`**.
  - Assign 1, 4, 5 to the variables: **`A, B, C`**.
  - Use expression to calculate the equation B * B + 4 *A * C and store the calculated value into a variable: **`temp`**
  - Calculate the square root of **`temp`** and save it into a variable: **`result`**.
  - ***Hint**: to calculate square root of a number X, use the following expression X**(1/2.0).*
  - Output the **`result`** with print() and you should see **`6.0`** has been printed to the console.

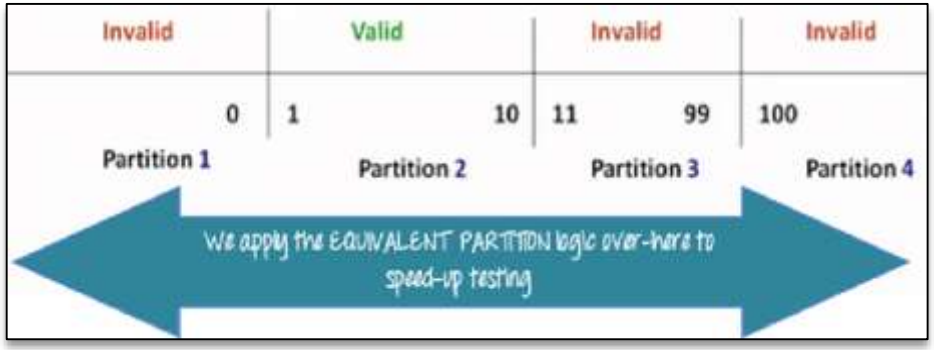
- The basic approach to design test cases depends on:
  - The software's specification only (**black box testing**).
    - Check whether the program outputs – i.e., 6.0.
  - The software's specification and the code that implements the software (**white box testing**).
    - Check whether the program outputs the correct result.
    - Check all implementation requirements are fulfilled.
      - In this example, check file name, variable names and values.

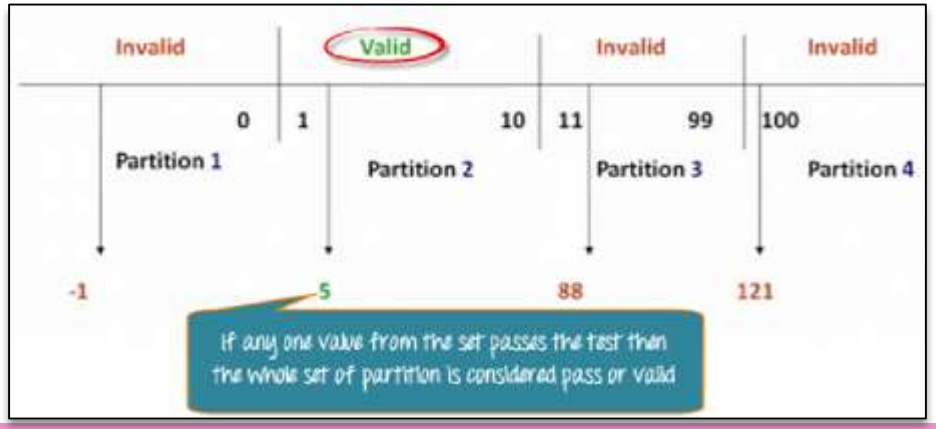
Test Case Design – what to consider?

- Test case design should consider many issues, e.g.:
  - Equivalence partitioning
  - Boundary value analysis
  - Statement coverage
  - Decision/condition coverage

Test Case Design - Equivalence partitioning

- Equivalence partitioning
  - Possible inputs could be very large and infinite, so testing is limited to a small subset of inputs.
    - Divide a set of test conditions into partitions that can be considered the same.
    - Then we take one value from each partition for testing. The hypothesis is, if one value of the partition pass, all other values will also pass. Likewise, if one condition in one partition fails, all numbers in this condition will fail.
- Example
  - Let's consider the behaviour of tickets in the Flight reservation application, while booking a new flight.
  - Ticket values 1 to 10 are considered valid & ticket is booked. While value 11 to 99 are considered invalid for reservation and error message will appear, **"Only ten tickets may be ordered at one time."**





Test Case Design - Boundary value analysis

- Boundary value analysis
  - Test cases are designed based on the boundary values between the partitions identified during the previous Equivalence partitioning test.
    - select the input from the boundary values
    - select inputs that produce output boundary values

- Example

```
 9  i = int(input("Enter a number: "))
10
11  if i > 5:
12      print("More than 6.")
13  else:
14      print("Less than 6.")
```

```
Enter a number: 7
More than 6.
```

```
Enter a number: 6
More than 6.
```

```
Enter a number: 5
Less than 6.
```

- Note. If you consider "equivalence partitioning", you could also test the numbers such as -2, 0, 100…

Test Case Design - statement coverage

- Code coverage: how completely test cases execute the code of the software system.
- Attempt should be made so that test cases:
  - Enter and exit every module/methods/procedures.
  - Execute every line of code.
  - Follow every logic and decision path through the software.


```
if __name__ == "__main__":
    print(add(1, 3))
    print(subtract(10, 4))
    print(multiply(2, 4))
    print(divide(5, 3))
    display_menu()
    run_calculator()
```

Have you tested the function get_input()?

Test Case Design - decision/condition coverage

- Decision/condition coverage
  - every condition in a decision in the program has taken all possible outcomes at least once
  - every decision in the program has taken all possible outcomes at least once.

- What do we need to test?
  - 90, 99, 100
  - 80, 89
  - 70,79
  - 60,69
  - 59,40,20

```
def find_grade():
    score = int(input("Enter 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")
```

Test case design: Test Table

Programme specification

- A programmer has developed a software to allow users to view, create, and edit contacts and contact lists.
- Once the programme starts, it loads all existing contacts, and then view a basic menu as shown below:

```
3 records have been loaded

 ===============
 enter for option
  =    ==========
  v    view contacts_list
  a    add contact
  d    delete contact
  q    quit

your option:
```

Test cases TC1

Test Case ID	Software Feature	Steps	Expected	Actual	Passed /Failed
TC1	View contact list	1. User to run the programme. 2. Main menu to appear with 4 options: 3. User to input option: v enter for option = ========== v view contacts_list a add contact d delete contact q quit your option: v	A list of already existing contacts will appear	A list of existed contacts will appear 1 Alice 01213334444 2 Bob 3467892 3 Sandra 0122011999 press enter to continue	**Passed**

Test cases TC2

Test Case ID	Software Feature	Steps	Expected	Actual	Passed /Failed

TC2	Add a new contact	1. User to run the programme.
		2. Main menu to appear with 4 options:
		3. User to input option: a
		`3 records have been loaded`
		`enter for option`
		`= ==========`
		`v view contacts_list`
		`a add contact`
		`d delete contact`
		`q quit`
		`your option: a`
		4. The system will prompt the user to enter the name and phone number of the new contact
		`enter name : Shadi`
		`enter phone number : 071234567`
			A new contact should be added to the list, and the user should be able to view once the option view contacts list is selected	A new contact has been added to the system, and is visible in the list of contacts
				`your option: v`
				`1 Alice 01213334444`
				`2 Bob 3467892`
				`3 Sandra 0122011999`
				`4 Shadi 71234567`
					Passed

Test cases TC3

Test Case ID	Software Feature	Steps	Expected	Actual	Passed /Failed

TC3	Deleting existing contact	1. User to run the programme.
		2. A main menu to appear with 4 options:
		3. User to input option: d
		`= ========== v view contacts_list a add contact d delete contact q quit your option: d`
		4. The user will be asked for the index of the contact to be deleted.
		`enter index of contact to delete 1`
		5. This should delete Alice with her Phone No
			Alice and her phone number to be deleted from the system.	Alice and her relevant Phone No has been deleted from the system.
				This is apparent when selecting the v option to list all available contacts.
				`your option: v 1 Bob 3467892 2 Sandra 0122011999 3 Shadi 71234567`
					Passed

Test cases TC4

Test Case ID	Software Feature	Steps	Expected	Actual	Passed /Failed

TC4	Deleting non-existing contact	1. User to run the programme.
		2. A main menu to appear with 4 options:
		3. User to input option: d
		`3 records have been loaded`
		`enter for option`
		`= ==========`
		`v view contacts_list`
		`a add contact`
		`d delete contact`
		`q quit`
		`your option: d`
		4. The user will be asked to provide the index of the contact to be deleted.
		`enter index of contact to delete 10`
		5. Although there are only 3 contacts, the user asked for contact with **index 10** to be deleted. This contact does not exist.
			User should get a warning message "Invalid index number" and should be sked to re-enter a valid index number ranging between 1 and 3	Software to crash: IndexError: list assignment index out of range	**Failed**

Test cases TC5

Test Case ID	Software Feature	Steps	Expected	Actual	Passed /Failed
TC5	Quit the system	1. User to run the programme. 2. Main menu to appear with 4 options: 3. User to input option: q to exit the system. 3 records have been loaded enter for option = ========== v view contacts_list a add contact d delete contact q quit your option: q	the system shutdown with no issues	The system has quitted with no problems 3 records have been loaded enter for option = ========== v view contacts_list a add contact d delete contact q quit your option: q In [2]:	**Passed**