

COM4013

Introduction to Software Development

- Week 5
- Recap
- Nested For Loops
- Debugger (still to be added)
- Review

• *Umar Arif – u.arif@leedstrinity.ac.uk*



Recap

- What's the difference between these while loops?

```
while ( True ):  
    print( "Hello" )
```

```
#-----  
#
```

```
IsPrintingHello = True
```

```
while ( IsPrintingHello ):  
    print( "Hello" )
```

```
#-----  
#
```

```
IsPrintingHello = True
```

```
while ( IsPrintingHello == True ):  
    print( "Hello" )
```

Try-Except Blocks

- We can use a `try-except` block to ***gracefully handle*** errors
- The code below handles a `ValueError` when the user enters a string instead of an integer below

```
While (True):  
    try:  
        userNumber = int(input("Enter a number > 5: "))  
  
        if (userNumber > 5):  
            print(f"You entered a  
            valid number: {userNumber}")  
            break # Needed to stop looping forever and  
            ever  
        else:  
            print("Invalid input. Please enter a  
            number.")  
    except ValueError:  
        print("Invalid input. Please enter a number.")
```


Where might we use a "for-loop"

```
for i in range(1, 11):  
    print(i)
```

- The range function works the same as before so we're counting from 1 (inclusive) to 11 (exclusive)
 - Exclusive in that the loop stops counting at 11
 - Meaning that this for loop will print the numbers 1 to 10
- Looping a limited amount of time
 - Unlike while loops we do not need to define a counting variable as it is in-built
 - Which part of the for loop defines the counting/loop variable? What is its name?
 - Passwords (get 3 attempts before computer locks)

A Single 'for' Loop Example

What does this for loop do?

```
for i in range(0, 20):  
    print( "X", end="" )
```

- It prints a horizontal line of twenty "X" characters
 - XXXXXXXXXXXXXXXXXXXX
- Note that adding end="" to the print statement means that the next line does not appear on its own line after displaying its text, so each 'X' appears immediately to the right of the last
- Using print alone (without the end="") would result in a vertical line of X's
- Also note the use of the variable name 'i'
- This isn't a very descriptive name, but for hard-to-name loops there is a convention of using variable names i, j, k etc.
 - This is the case in my style guide (please keep this in mind when doing the assignment)

A Nested Loop Example

What does Nested mean? Also, how does it work?

```
for i in range(0, 10): # Repeats the inner loop
    for j in range(0, 20): # Draws one line
        print("X", end="")

    print("\n", end="") # EXTRA PRINT - step to
                        # next line
```

So, what does this loop do?

- On the inside is the loop to draw a single line of 20 X's
- But that loop is itself repeated 10 times, so 10 lines are drawn
- The **EXTRA PRINT** statement is important as it moves the printing to the next line after each line of the 20 X's is drawn





Nested Loop Iterations

How many times will the inner code execute?

```
for i in range(0, 10):  
    for j in range(0, 20):  
        # Inner code goes here
```

- The inner loop counts 7 times (0 to 6, notice it says less than 7)
- The outer loop repeats the inner loop 5 times
- So, the very inner line will execute $5 \times 7 = 35$ times
- We call each time round a loop an ***iteration***
- So, the inner code has a total of 35 iterations

Uses of Nested Loops

- Nested loops are commonly used for:
 - Working with data arranged in a grid
 - We just saw a grid of X's
 - Working with lists within lists
 - E.g., We have 180 students in one list, and each student has 6 module marks saved in another list:

```
for student in range(180):  
    for studentMark in range(6):  
        # Do something with the marks of a particular  
        # student
```

- We will work with examples like this one later in the module
- What is the default start value of the for loop?



Syntax Error & Bugs

A **syntax error** is when your code is not valid Python

- You have broken the rules of the language in some way
- Syntax errors are shown in Visual Studio with red underlines

A **bug** is an error that does not break the rules of Python, but causes incorrect behaviour when the program is executed

- Bugs come in many varieties:
- Infinite loops, incorrect conditions, incorrect formulas, crash bugs...
- A common bug includes mixing up the "and" and "or" keywords



Debugging

- Debugging is the process of carefully analysing a program to identify the causes of bugs
 - Don't be disheartened if you get bugs in your programs!
 - Professional software developers with years of experience spend time everyday debugging
 - Bugs & debugging are a normal part of the programming process
- We use a special tool called a **debugger** to help identify bug
 - Allows you to step slowly though the program
 - Shows the variables and results at every stage
 - So, we can look for anomalies
- Python's in-built debugger is not very good, so I won't be spending long on it.

PDB Debugger

| Command | Description |
|---------------|--|
| help or h | Get help on PDB commands. |
| list or l | List the source code around the current line. |
| break or b | Set breakpoints in your code. |
| continue or c | Resume execution until the next breakpoint. |
| step or s | Step through code line by line. |
| next or n | Continue to the next line in the current function. |
| up | Move up the call stack. |
| down | Move down the call stack. |
| p or pp | Evaluate and print expressions. |
| args | Display function arguments. |
| locals | Display local variables. |
| globals | Display global variables. |
| where or w | Show the current call stack. |
| interact | Start an interactive Python shell. |



PDB Debugger Example

```
import pdb

for i in range(0, 10):
    pdb.set_trace()
    for j in range(0, 20):
        print("X", end="")

    print("\n", end="")
```

--Output--

```
ipdb> c XXXXXXXXXXXXXXXXXXXXXXXX > # type c to see the
next loop or q to quit (will result in error)
```

When running this code, we can use the c or continue command to see each loop of the inner for loop line.

It is not great thought, and you may find that there are other debuggers that you find to be more intuitive.



Review

- We have now covered the core techniques used by most programming languages
 - There is plenty more material of course
- Now is a good time to pause and review each of the techniques covered so far
 - In this week's lab there is a practice section that also reviews what we've covered

Review: Variable Declaration/Initialisation

Variable **declaration**

- `numberStudents = None # declared to nothing`

Variable **declaration** and **initialisation**

- `numberStudents = 180`

Different **types** of **variable**

- `userHeight = 178.5`
- `userName = "Joe"`
- `userInitial = 'J' # What type of variable is this`
- `isEnrolled = True`

- **Good** and **Bad** variable names

- `NumberSides = 4 # Good variable name (and this is a comment)`
- `thingy = "Thing" # Bad variable name, not clear what the variable will do`

Review: Variable Assignment, Expressions

Variable **assignment**

- Putting a value in a variable after the declaration
- `numberSides = 6`

Assignment using an **expression** (a calculation of some kind)

- `wallArea = wallWidth * wallHeight`

Updating a variable

- `numberFiles = numberFiles + 1` `||` `score = score * 2`

Shorthand versions of the above

- `numberFiles += 1`
- `score *= 2`

Using **brackets** to make sure **expressions** give the correct result

- `fahrenheit = (9.0 / 5.0) * celsius`

Review: Displaying Text

Displaying text, each time on a new line:

- `print("Hello World")`
- `print("Hello Again")`

--Output--

Hello World

Hello Again

Displaying text, each time on a new line:

- `print("1234", end="")`
- `print("5678", end="")`

--Output--

12345678

Displaying text and variables, place variables in f strings for printing, where they are replaced with the true variable values.

- `print(f"You are a {type} and your health is {health}")`

--Output--

You are a warrior, and your health is 120

Review: Inputting Text

Reading a line of text (name) typed in by the user into a string variable

- `print("what is your name?: ")`
- `userInput = input()`

Displaying a message and input all on the same line. Printing on the same line on the console too (do it this way)

- `userInput = input("Enter your name: ")`
- `print(userInput, end="")`

--Output--

`Enter your name: Oliver`

Converting input text to lower case (I will not be using caps lock when testing assignments)

- `userInput = input("Enter your name: ").lower()`

Review: Converting Text to Numbers

Read text from users as a string, and then convert (**cast**) it to an integer

- `userAge = int(input("Enter your age: "))`

Reading a floating-point number from the user

- `userNumber = float(input("Enter a number between 0 and 1: "))`

Note that it is important to not cast a float to an integer as it will always be rounded down. Use the round function instead

- `userNumber = float(input("Enter a number between 0 and 1: "))`
- `roundedNumber = round(userNumber, 2)`

You can round floating-point numbers in the print statement themselves

- `print(f"You have £{userNumber:.2f}")`

Removing leading and trailing 0's (this works for String inputs **ONLY**)

- `userNumber = float(input("Enter a number between 0 and 1: ")).strip()`

Review: Conditions

Simple conditions (an expression that is `True` or `False`)

- `numberSides > 4`
- `characterType == "warrior" # Use double == not single = in conditions`

Conditions combined with '`and`' keyword and the '`or`' keyword

- `age >= 20 and age <= 29`
- `name == "Joe" or name == "Jane"`

Avoid conditions that are impossible

- `numberFiles < 10 and numberFiles > 20 # Why is this impossible?`

Conditions are not used on their own

- They can be put into Boolean variable
- These can be used in if, while and for loops (see next slides)
- `isTeenager = (age >= 13 and age <= 19)`

Review: 'if' Statements

if statement

```
if (name == "Jim"):  
    print("Hi Jim, you owe me some money...")
```

if-else statement

```
if (guess == "tomatoes"):  
    print("Well done, you guessed correctly!")  
else:  
    print("Sorry, that's not correct...")
```

Review: 'if' Statements

Several `if-else` statements linked together

```
if (characterType == "warrior"):
    strength = 150 # Strong, but poor magic
    magic    = 50
elif (characterType == "wizard"):
    strength = 50  # Good at magic, but weak
    magic    = 150
else: # All other character types (e.g., scout, thief, etc.)
    strength = 100 # Balanced
    magic    = 100
```

Review 'while' and 'for' Loops

while loop :-

```
yesNo = input("Enter 'y' or 'n': ").lower()
```

```
while (yesNo != 'y' and yesNo != 'n'):
```

```
    print("Only enter 'y' or 'n': ")
```

```
    yesNo = input("Enter 'y' or 'n': ").lower()
```

for loop :-

```
for count in range(1, 11):
```

```
    squared = count * count
```

```
    print(f"{squared}, ", end="")
```