COM4013 Introduction to Software Development

- Week 4
- More Loops
- User Input Validation

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

Recap

- Thus far, we have been over variables, intialisation, casting, expressions, input, and more
  - `name = "Umar"`
  - `age = 18`
  - `pounds = 14.5`
  - `letter = 'C'`
  - `isAlive = True`
  - `z = 3 + 4j`
- Operators, precedence, and parenthesis
  - `+ - / *`
  - `+= -= /= *=`
- If statements
  - `if (cond):`
  - `elif (cond):`
  - `else:`
- While statements
  - while (cond):

```
name = input("What is your name?")
print(f"Your name is {name}")
print("Your name is: ", name)
```

What's the difference between the prints?

```
age = int(input("What is your age?\n")
print(f"Your name is {name}")
```

What does the \n do?

What does \t and \s do?

```
moneyOwed = float(input("How much money
does Jim owe me?")
print(f"Jim owes {moneyOwed}")
```

What happens when we cast a float to an int?

```
isDead = True

If (not isDead):
    print("He's dead! :)>")
else:
    print("Dig his grave... jk")
```

What's printed out when we call this code?

```
elif ( isDead ):
    print("Tf... You're
    redundant code")
```

What's printed out when we add this to the above code? Where would we add this?

Operator Precedence

1	()	Brackets
2	not, +, -	Logical Not, making a value positive or negative
3	* / %	Multiply, divide, modulus (remainder)
4	+ -	Addition, subtraction
5	< > <= >=	Greater & less-than comparisons
6	== !=	Equal & not-equal comparisons
7	and	Logical And
8	or	Logical Or
9	= += -= *= /= etc.	Assignment and assignment operators

How do we get around having to think about precedence?

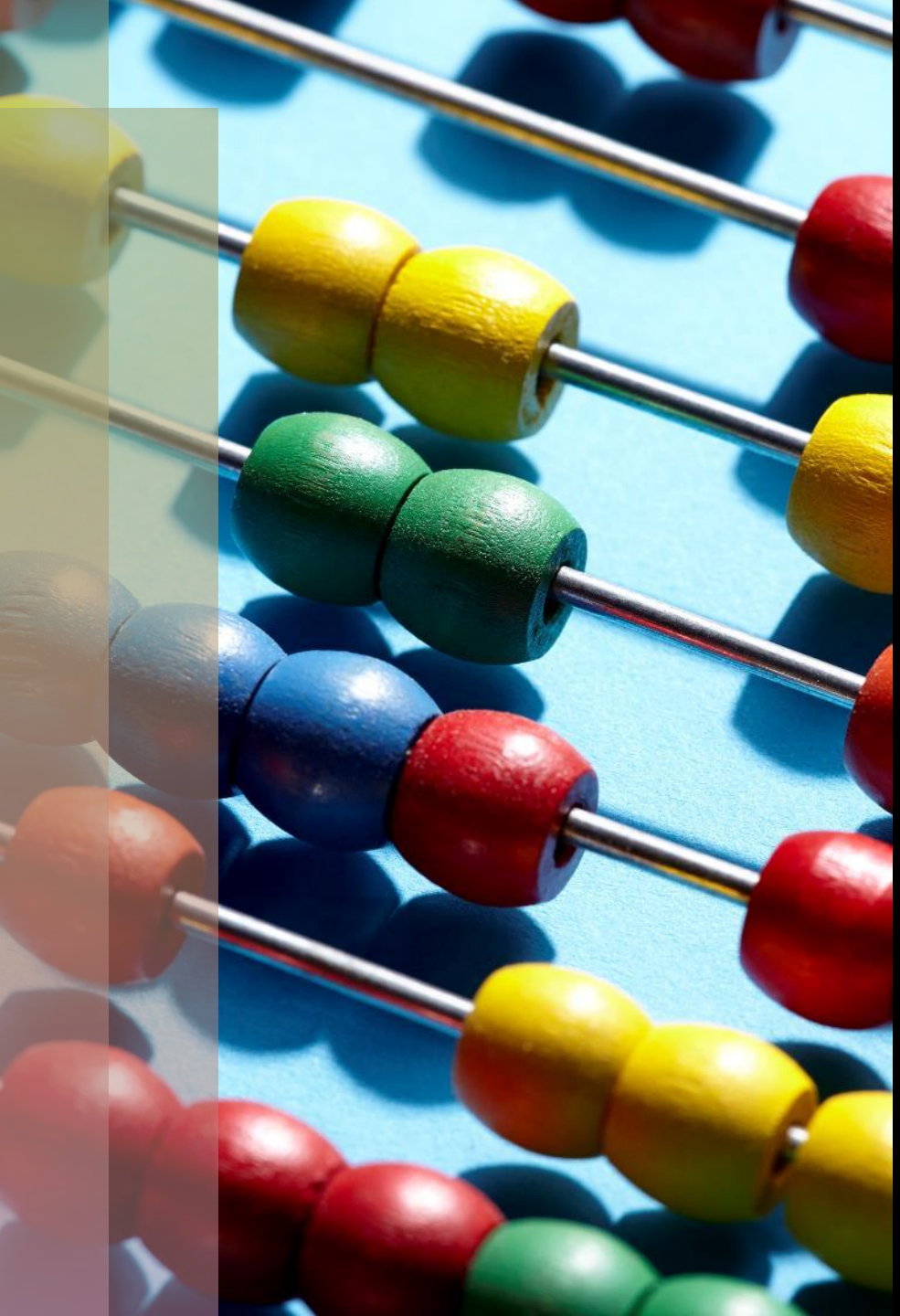
If we wrote an if statements, how could we make it more efficient?

Reminder: the ‘while’ Loop

- Last week we saw a while loop that counted from 1 to 10:

```
counter = 1
while (counter <= 10):    # Do we need the parenthesis
    print( counter );     # Will this line work?
    counter++
```

- Counting in loops is very common, e.g.
- Counting through the files in a folder
- Counting through a list of student records
- Counting the number of attempts in a guessing game
- Note: we don’t count time (e.g., seconds) like this
  - Computers are fast, this loop will finish in < millionth of a second
  - You will see methods for timing later in your courses.



The Parts of a Counting Loop

We can identify three key parts in a counting ‘while’ loop:

```
counter = 1 [A]
while (counter <= 10 [B]):
    # Code to be repeated
    counter++ [C]
```

- [A] Declare and initialise a counting variable
  - The variable can be declared earlier, but at least give it a starting value
- [B] Use a condition to determine how long to keep looping
  - Usually compare the counting variable with some ending value
- [C] Step the counter
  - Usually step by 1, but we can step by different values

The ‘for’ Loop

- The for loop is simply a shorthand for this kind of while loop:

```
for counter in range(1, 11):
    print(counter)
    counter += 1
```

- NOTE: The 1 is inclusive, so the for loop will begin counting from 1 but the 11 is exclusive so the for loop will end counting at 10.

- This does the exact same as the code on the last slide

- Easy to read this and see that the for loop counts from 1 to 10...
  - ...and the code inside will repeat exactly 10 times

The ‘for’ Loop: Part 2

```
for i in range(1, 11, 2):
    print(i)
```

- The range function works the same as before so we're counting from 1 (inclusive) to 11 (exclusive)
  - Exclusive that it stops at 11 and prints up to 11
- Note that there is a 2 after the for stop code
  - Increments the loop by 2 instead of the default which is one.

```
for variable in range(start, stop, step):
```

  - What are we printing here?

Examples of ‘for’ Loops

- A loop counting from 0 to 99:

```
for i in range(100):
    # Code to be repeated
```

- Notice how we're only specifying the **stop** value here
- If we only provide a single value, Python will start counting from 0
- For sanity purposes it makes sense to declare both the **start** and **stop** aspects of the for loop

```
for i in range(10, 0, -1):
    print(i)
```

- What is the above code doing?

‘while’ or ‘for’ Loop?

- If you are counting, especially a fixed number of things, then use a for loop
  - Either you know the exact number (10 files, 180 student records)
  - ...or the user will supply the number (example on last slide)

- If the number of loops needed is unknown, then a while loop can be clearer
  - Last week's guessing game for instance works well, as we don’t know how many guesses will be needed:

```
userGuess = input("Guess the word: ");        # User’s first guess

while (userGuess not "tomato"):               # Loops while errors remain
    userGuess = input("Wrong, guess again:") # User has another guess

print("You win!") # Finally guesses "tomato" correctly
```

Looping Zero times

- In this example, if the user guesses correctly first time, then the code in the loop block is not executed at all
  - Program skips directly to the “win” message:

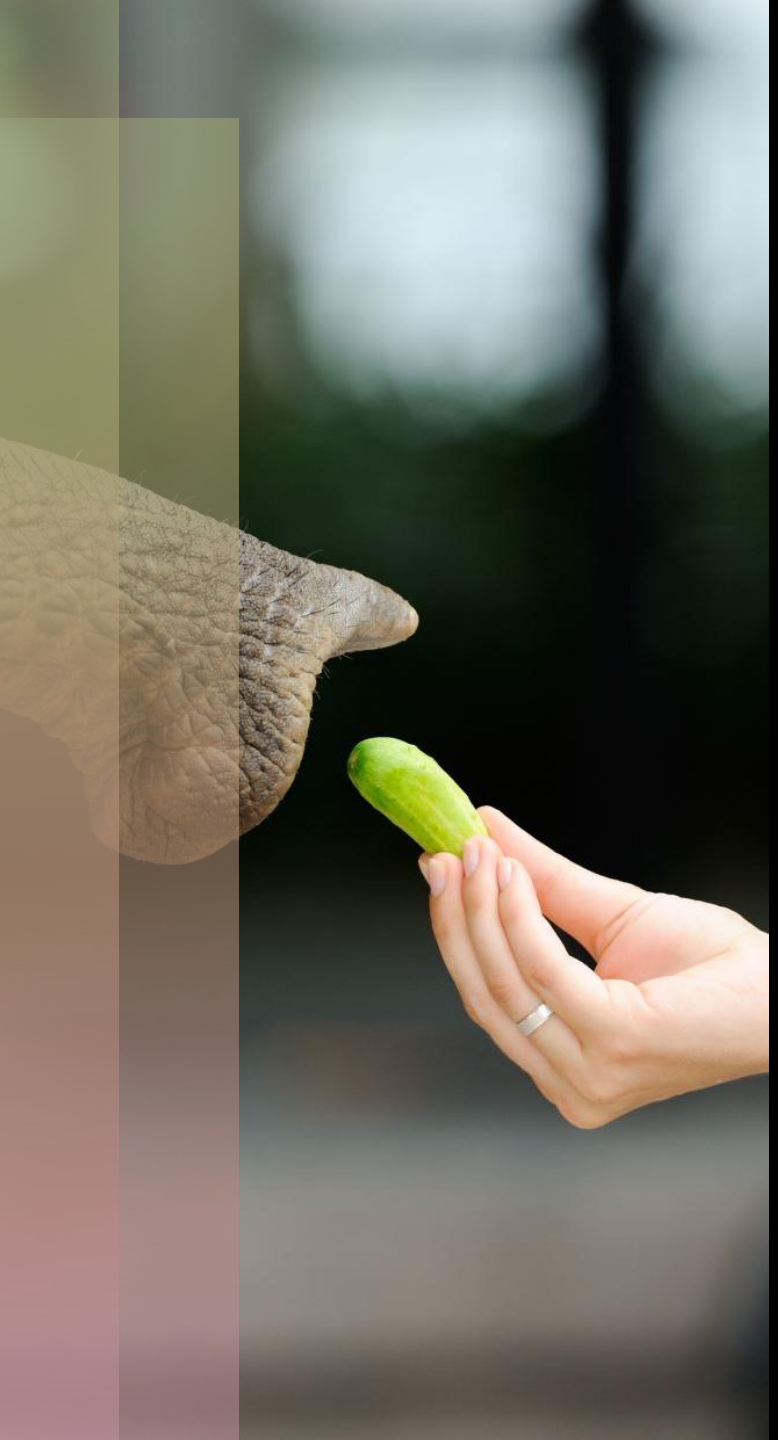
```
userGuess = input("Guess the word: ");        # User’s first guess

while (userGuess != "tomato"):                # Loops while errors remain
    userGuess = input("Wrong, guess again:") # User has another guess

print("You win!") # Finally guesses "tomato" correctly
```

- Sometimes we want a loop where we can guarantee it will execute the loop block ***at least one time***.

The ‘do-while’ Variant

- Do-while loops don't exist in Python (like many other things...).
- Last week I mentioned that when we want to validate a user's input, we will first ask them to input something specific, followed by a while loop (which checks for a specific response).
- If we do not get the input, we're looking for then we return an incorrect message and provide a second chance for an input.

```
while (True):
    # Code to be executed at least once
    user_input = input("Do you want to continue? (yes/no): ")
    if ( user_input.lower() == "no" ):
        # Resolve input here
        Break
    # Could have other elifs, and else here jic they never input
    what you want  them to.
```



User Input Validation

- Most of our code examples have had some user input
- So far, we have assumed their input will be correct:
  - The user will make no mistakes and uses the format we want

```
Welcome to Dungeon Quest
What kind of adventurer are you, "Warrior" or "Scout"?
Bus Driver
```

- A well-written program will:
  - Identify when the user has entered something invalid
  - Warn the user with a helpful message
  - Allow the user to enter the information again
  - Repeat this process as long as necessary

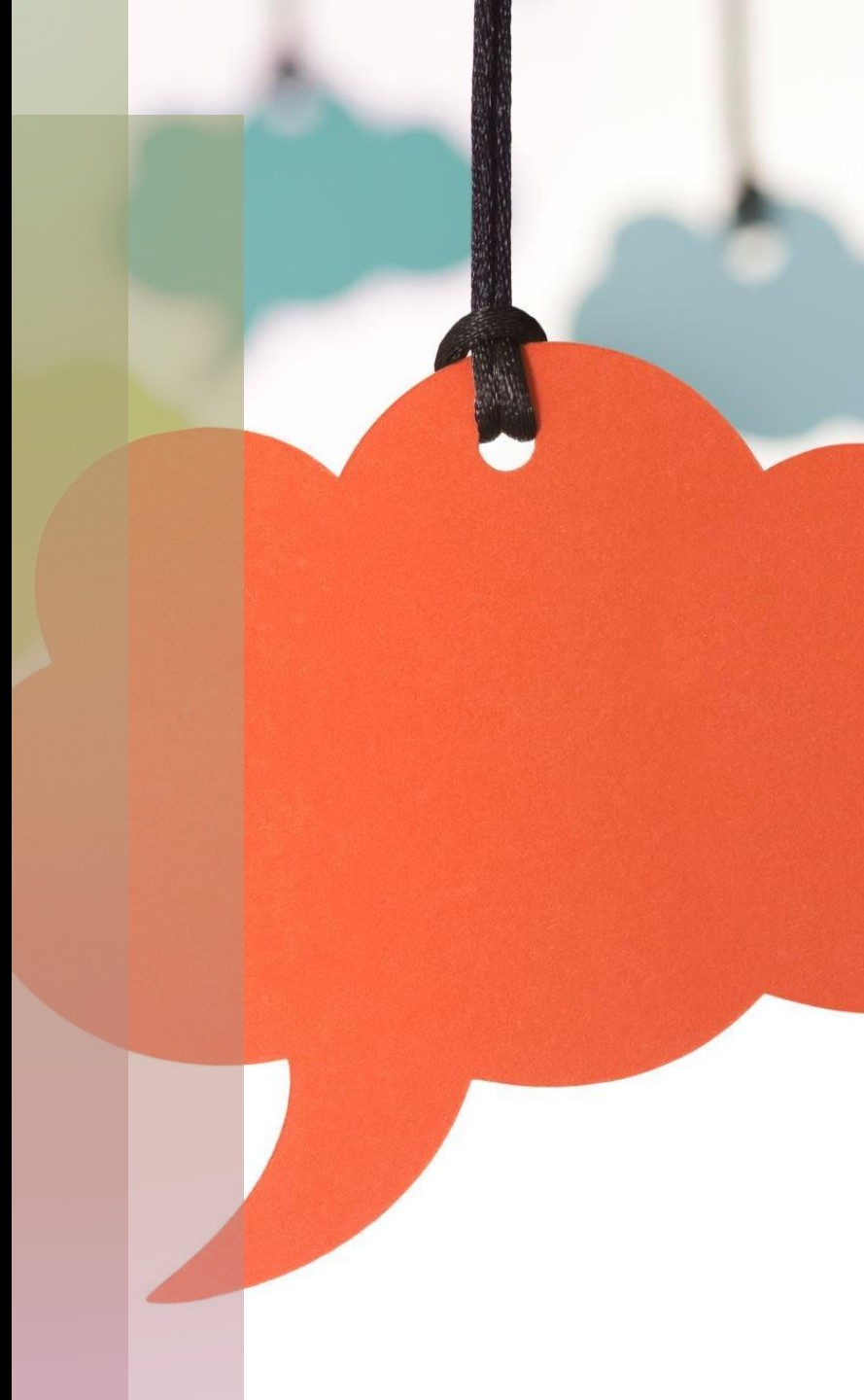
User Input Validation in Action

Welcome to Dungeon Quest
What kind of adventurer are you, 'Warrior' or 'Scout'?
Bus Driver

I'm sorry, a 'Bus Driver' is unsuitable for this quest.
Please choose only 'Warrior' or 'Scout':
Ballerina

A 'Ballerina' would not last long in this perilous dungeon.
Please choose only 'Warrior' or 'Scout':
Warrior

Welcome brave Warrior...



Text Validation

- We can validate a text string with something very similar to the earlier guessing game code:

```
characterType = input("Enter Warrior or Scout: ") [A]

while (characterType != "Warrior" and characterType != "Scout"): [B]
    print(f"Sorry you cannot be a {characterType}") [C]
    characterType = input("Please only enter Warrior or Scout: ") [D]

print(f"Welcome brave {characterType}")
```

- Give the user a first attempt to input [A]
- If they make a mistake [B] then enter a loop which repeatedly:
- Displays an error message [C] and asks again for the user input [D]

Upper- and Lower-Case

- What if the user types "WARRIOR", or "warrior" (or even "wArRiOr")?
  - Not really a mistake, the program should accept this
  - The Python string type has two methods to help here:

- upper(): converts any lower-case characters to upper case
- lower(): similar, converting upper case to lower case

```
testText = "THE 5 WarriorS"
lowerText = testText.lower()
print( lowerText )
```

- Will output all in lower case:
  - the 5 warriors

User Input: Converting to Lower Case

- It is often a good idea to convert all user input text to lower case
- The user can type using any kind of case
- We only need to compare against one version (e.g., "warrior")
- Very easy to do this using lower()

```
characterType = input("Enter Warrior or Scout: ").lower()
```

- Note how it's all done on the same line. We can chain methods together. `.lower().strip().upper().capitalize()`

Validating Numeric Input: Ranges

- The previous examples considered text input
- For numeric input there are some similarities
  - We often want check that a number is in a ***valid range***
  - Do this using a while loop like the earlier example:

```
userNumber = int( input( "Enter a number from 1 to 100: " ))
while (userNumber < 1 or userNumber > 100):
    userNumber = int( input( "Please only enter a number
    from 1 to 100" ))
```

- As usual be careful to choose the correct condition
- What would happen if we used and in this example?

Try-Except Blocks

- We use a try-except block to ***gracefully handle*** errors when attempt converting user input to an integer.
- If successful, isNumber is set to True.
- If the conversion fails due to a ValueError (e.g., user enters a non-numeric value), isNumber is set to False.

```
try:
    userNumber = int(input("Enter a number: "))
    isNumber = True
except ValueError:
    isNumber = False

if (isNumber):
    print(f"You entered a valid number:
    {userNumber}")
else:
    print("Invalid input. Please enter a number.")
```

Avoid Infinite Loops... Break free!

```
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.")
```

Break statements are great, and can be used in for loops too... I will talk about this more in a future session.

Summary

In today's lecture we have covered:

- The `for` loop, used for counting things
  - Just a shorthand for a `while` loop
- The `do-while` loop variant (doesn't exist but works similarly)
  - When you want to guarantee that the loop will execute at least once
- Using `while` loops to check that user input is valid
  - Checking text input against valid choices
  - Checking numbers are in the correct range
  - Display a helpful error message and make the user try again
- Using `.lower()` to make all text input lower case for simplicity
- `try-except` block example – do not need to use this