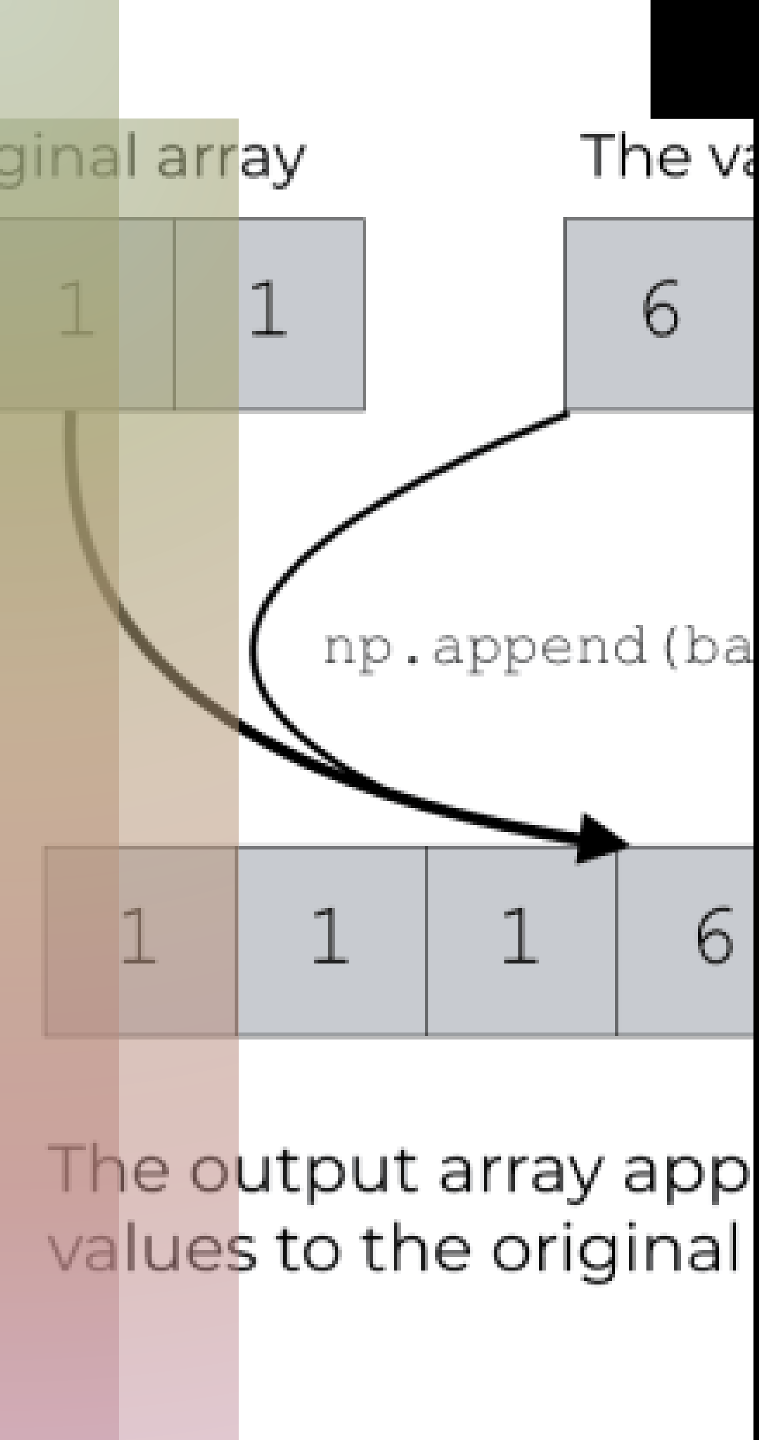
COM4013 Introduction to Software Development

- Week 8
- Functions and Classes

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



Write File Modes: Recap

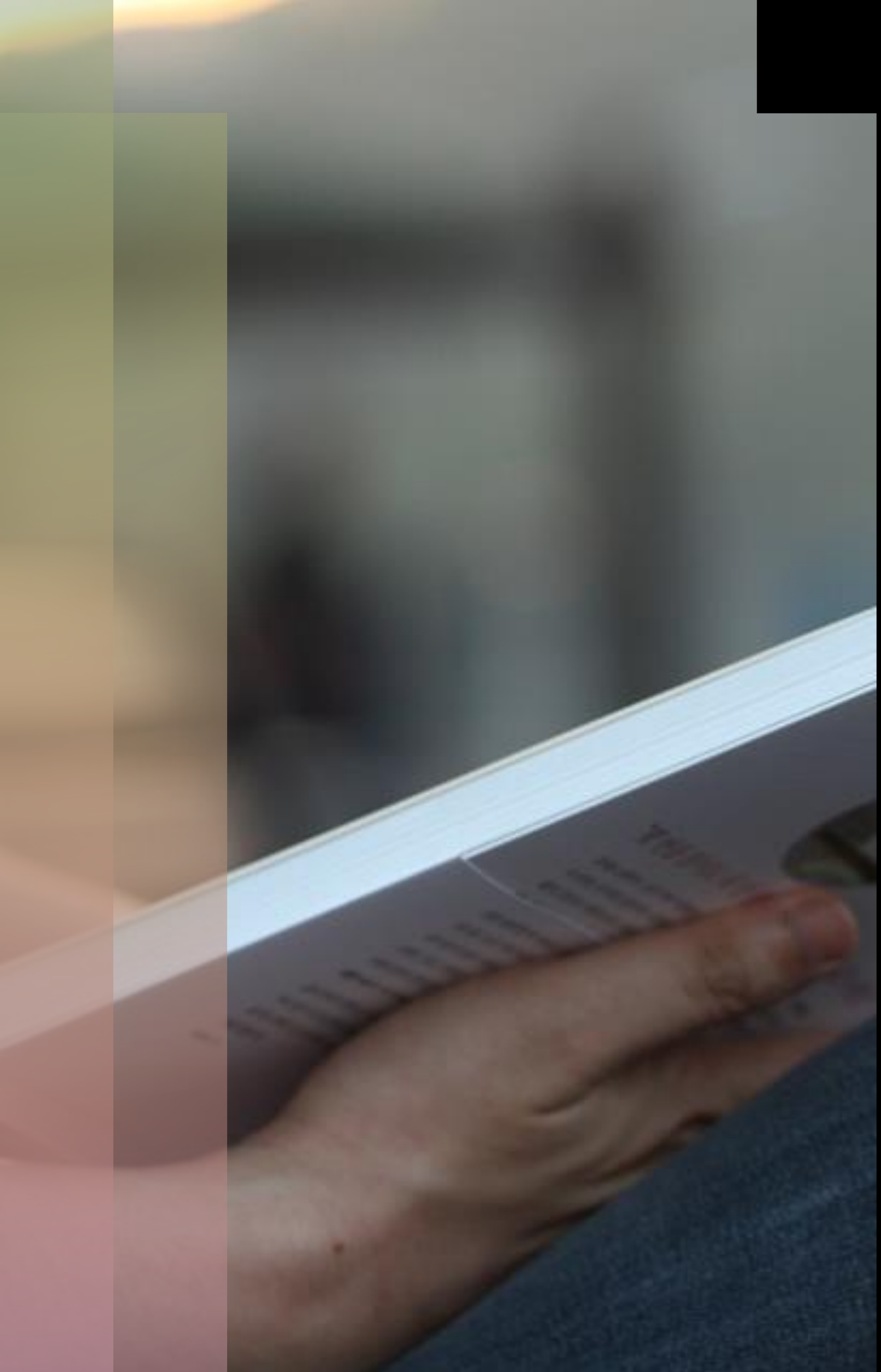
The `open()` function provides many modes for writing. There is the write `"w"` mode which creates and/or rewrites over the contents of a file

There also exists an append `"a"` mode which creates and opens a file for writing but does not overwrite pre-existing content

```
# Open a file for writing (creates it if non-existing)
with open( "test.txt", "w" ) as file:
    file.write( "Hello World\n" )
    file.write( "This is very basic" )

# Adds the line at the end of the test.txt file
with open( "test.txt", "a" ) as file:
    file.write( "Hello World\n" )
```

What will the resultant file look like?



Reading a Text File: Recap

Reading Type 1:

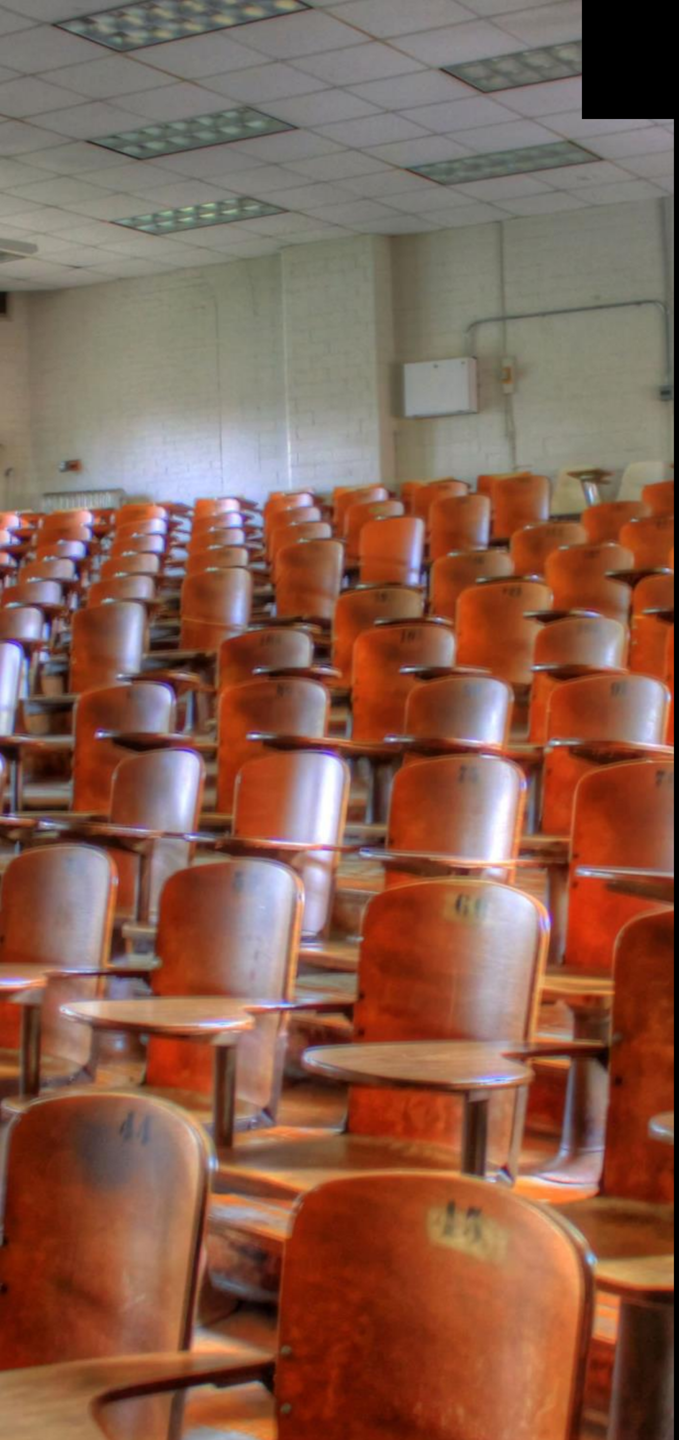
```
# What does this one do?
variable = file.read()
```

Reading Type 2:

```
# What does this one do?
variable = file.readlines()
```

Reading Type 3:

```
# What does this one do?
variable = file.readline()
```



Functions Concept: Recap

A function is a block of code that performs a specific task

- We define a **function** by giving it a name and writing its code
- We can **call** a function's code anytime in other parts of our code
    - To ‘call’ simply means to execute the code in a function

```
def PrintName():
    name = input("Please enter your name")
    print(f"Your name is... {name}")
```

NOTE: We are **NOT** passing a parameter to the function here BUT we could...

```
name = input("Please enter your name")
def WaitForEnter(firstname):
    print(f"Your name is... {firstname}")
```

NOTE: The **parameter** named in the function **definition DO NOT** need to match for functions to work

Function: Return Nothing

We've just seen a function example of when you can **define** a function.

- it doesn’t return an “answer”, just performs some task

However, you can still use the return statement in a function

- As there is no 'answer' just write return on its own

```
def HorizontalLine(lineWidth):
    if lineWidth <= 0:
        return

    for i in range(lineWidth + 1):
        print("-", end="")

    print()

# What is the end, step, and start of this for
loop? How would we call this function?
```

Variable Scope and Functions

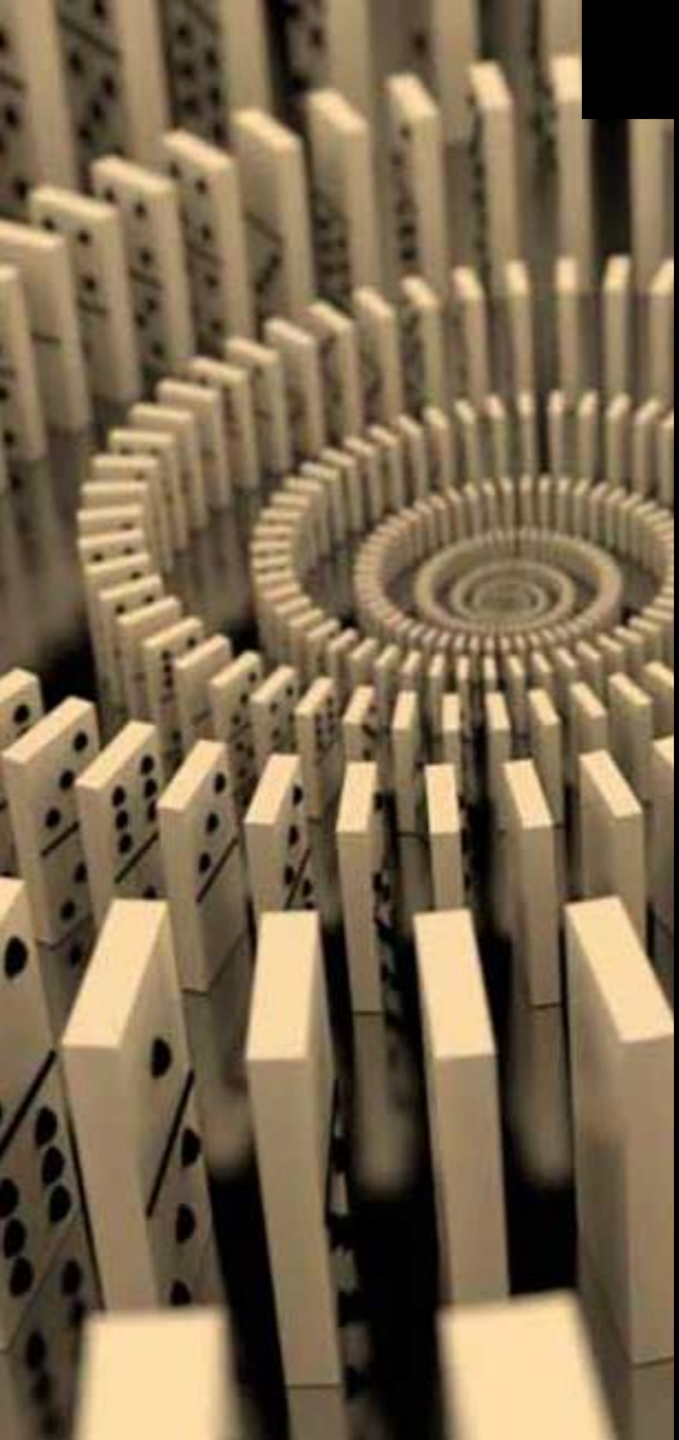
Variable scope also applies to functions:

```
def InputUserNumber( message ):
    userNumber = int(input(message))
    return userNumber
```

Variables declared as parameters or inside a function are only accessible within the function

However, we can explicitly return variables from functions and use those to modify variables outside of our functions

```
height1 = InputUserNumber("Enter height 1: ")
height2  = InputUserNumber("Enter height 1: ")
height3 = InputUserNumber("Enter height 1: ")
NoOfUsers   = 3
AverageHeight = (height1 + height2 + height3) /
noOfUsers
```



Literals

A **literal** is a specific value typed directly into the program:

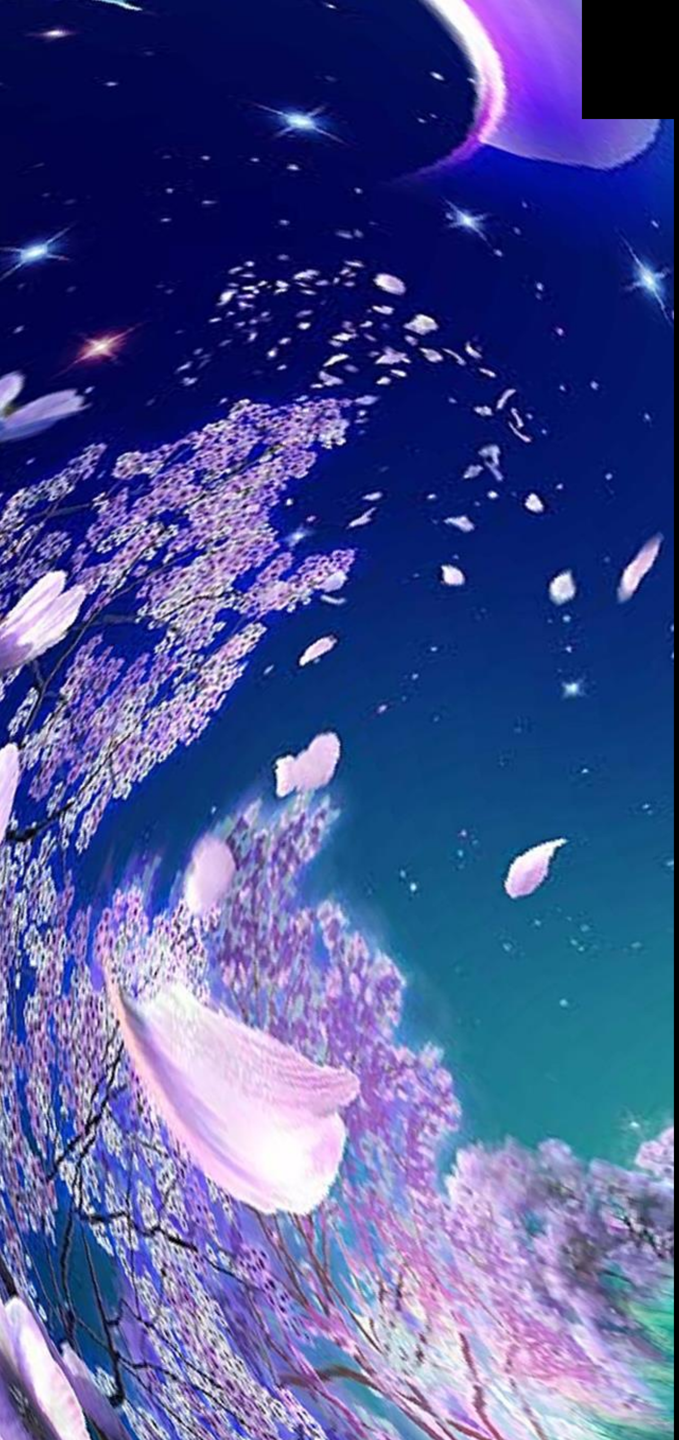
```
numberStudents = 180
characterType  = "Warrior"
```

- The numberStudents and characterType are variables
- The 180 is an **integer literal**
- The "Warrior" is a **string literal**

Literals occur frequently in programs

- How many literals are there in this code?

```
for i in range(0, 3, 1):
    print("-")
```



Magic Numbers

Why does this code use the literal value 80?

```
for i in range( 80 ):
    print("-")
```

- It's because the default width of a console window is 80 characters
  - So, this will draw a line across the full width of the window
- I know that, and maybe some of you do too, but many wouldn't

That 80 is an example of a "magic number"

- A literal value with a special meaning
- But because it's a literal the reader cannot see what the meaning is

- *Magic numbers are poor programming practice*

Functions and Parameters?

We've seen one idea already, put the code in a function which uses a parameter for the line width. The 80 is gone:

```
def HorizontalLine( lineWidth ):
    for i in range(lineWidth + 1):
        print("-")

    print()
```

- This is good style, but doesn't really solve the problem of magic numbers, because somewhere else we must write this:

```
HorizontalLine( 80 )
```

- We simply moved the magic number to the calling code...

Variables to Hold Magic Numbers

```
CONSOLE_WIDTH = 80

def HorizontalLine():
    for i in range(CONSOLE_WIDTH):
        print("-", end="")

    print()

def main():
    # Function call
    HorizontalLine()
```

You can declare variables above functions, outside of them. In this instance I've used **Snake Case** as generally variables that we declare and have no intention of changing are constants.

Unfortunately, Python has no constants as it is a **dynamic** programming language.

This is a good idea as we can declare the variable once and use it wherever it's needed.

Chaining 'if' Statements

We have seen “chains” of if-else statements to compare something against many possibilities:

```
name = input("Enter a name: ")

if name == "Jim":
    print("Hey Jim, you owe me some money...")
elif name == "Umar":
    print("Hi Umar, have you got my phone?")
else:
    print(f"Hello {name}")
```

Each condition is testing if name is equal == to something

Booleans, ‘and’, ‘or’

The **Boolean** type holds the value True or False

```
    giveDiscount = True
```

Conditions are expressions that give a Boolean result (True/False)

```
    playerScore > highscore
```

Use Boolean variables to hold the result of a conditions:

```
    newHighscore = playerScore > highscore # Evaluates to 1 or 0
```

Or use if, while, for statements to test conditions, for example:

```
    if (playerScore > hiscore)
```

Combine conditions with ‘and’ keyword and ‘or’ keyword:

```
     if (age >= 20 and age <= 29) # Test if age is in the
    twenties
```

Object Orientation Usefulness

The usefulness of Object-Oriented programming is as follow:

- Allows an approach called encapsulation, which means the creation of independent modules.
- Each class is treated as a black box. As the internal working of the class are hidden from the users and we can only access the classes member variables and methods using the public interface.
- If we were to provide the class to another person, all they would need to know is **what** it does, and not **how** it does it.
  - This is known as the **interface** to a class.
- You have been using classes all semester long. Think back to strings and all their useful ~~functions~~ **methods**.

```
name = "Umar"
print(type(name))

-- Output --
Output: <class 'str'>
```

Classes

A **class** is a user defined structure.

It is interesting (and very useful) because it allows you to define data types and methods (like functions) which are used all in one package.

Classes serve as a blueprint for creating **objects**. The define the properties (member variables) and the behaviours (methods) that objects of the class will have.

Object orientated programming use **objects** which are **instances** of a class.

Classes: How

To **define** a class, we can use the following syntax:

```python
class CSquare:
    mType   = None # Note the m prefix to member variables
    mLength = 0    # Read the style guide if you're confused
    mWidth  = 0    # You will be expected to follow it in A1

    # A constructor
    def __init__(self, type:str, length:float, width:float):
        self.mType   = type
        self.mLength = length
        self.mWidth  = width
```

NOTE: Nearly every class requires a constructor (I will come back to this this next week). When an **object** or an **instance** of the class is **declared** this constructor is our way of setting the member variables of the class.

NOTE: **self,** references the object itself (the object can call its own member variables).

Classes: Breakdown

```
[A]class [B]CClass:
    [C]mType = None # Note the m prefix to member variables

    [D]def __init__([E]self, [F]type:str):
        self.mType = type

    [G]def GetType(self): # Needs meaningful comments
        return self.mType  # Example of a getter
```

A. Use the class keyword to start defining a class – template for objects of said class
B. Name of the class will be used when declaring an instance of said class
   NOTE: The C prefix on the class name (read the style guide)...

C. We declare our member variables here. We can assign values to them here if we want to when a scenario arises that every instance of this class requires this variable.
D. We define the behaviour of the class using methods (they are like functions)
E. Pass **self** as a parameter for any methods that you need access to member variables
F. You can use type hints (`:str`) to make you methods/functions easier to read
G. We can define other methods too. I expect comments on all vars and class methods.

Using Classes

To call our CSquare class in main we do the following:

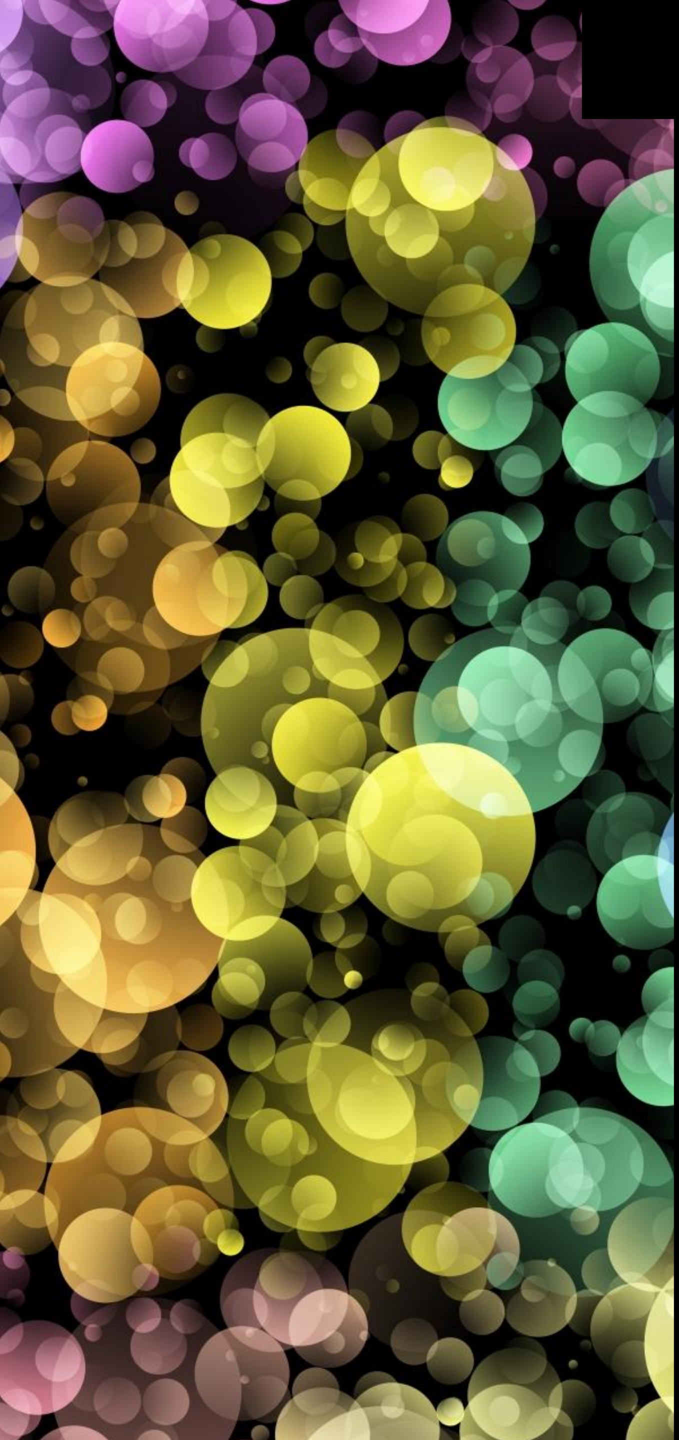
```
def main:
    square = CShape("square", 10, 12)
    type = square.GetType()
    print(type)
```

As you can see, the constructor of the class (on slide 14) defines the data you must provide for the class itself. If you do not provide the correct data in the correct order, then you will get an error

In python, we can dynamically add member variables to a class

```
    square.colour = "RED"
```

Doing this will make your code less readable. Hence, all member variables **MUST** be declared in the **class declaration** itself.



Dot Operator

In Python the dot (`.`) operator is used to access a classes member variables and its methods.

```
class CSquare:
    . . . # Constructor and member variables above

    # Change member variable values with Setter method
    # This is good practice
    def SetLength(self, length:float):
        self.mLength = length
```

Before we went over Getters which are methods that we use to get a variable from a class. Above is an example of a Setter which is used to change the value of a member variable after it has been declared

```
square.SetLength(14) # We use the dot operator to access
the class method here
```

Pass Keyword

In Python the pass keyword is used as a placeholder for eventual code.

```
def NumSquared(num:int): # I've used type hints here
    pass

def main():
    pass

if __name__ == "__main__": main()
```

Before we went over Getters which are methods that we use to get a variable from a class. Above is an example of a Setter which is used to change the value of a member variable after it has been declared

```
square.SetLength(14) # We use the dot operator here
```

Summary

We've covered quite a few topics today:

- Write and read
- Functions
- Variable scope - function scope for Python, block for other languages
- Magic numbers, readability, and good code practices
- Classes, member variables, constructor, dot operator, setter, and getters

Any questions?