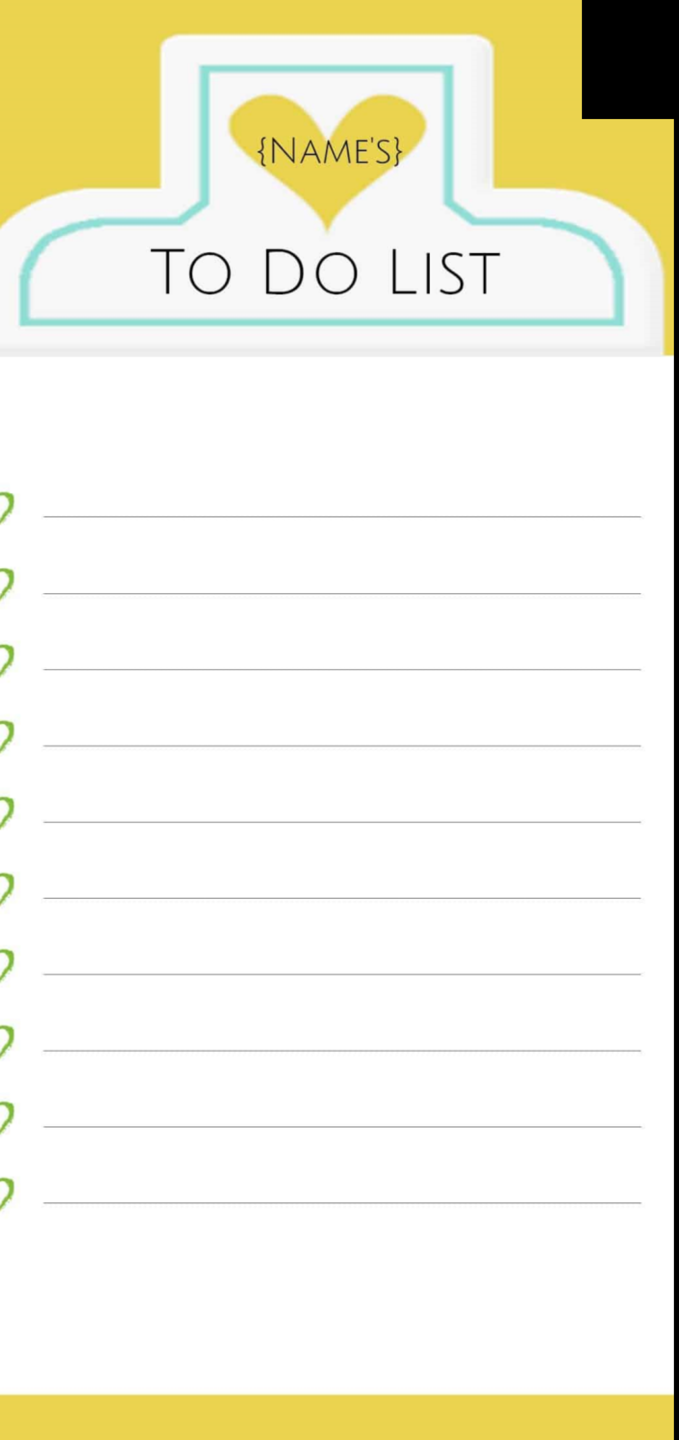
COM4013
Introduction to Software Development

- Week 6
- Lists, I/O, and Functions

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



List: Concept: Recap

A **List** is a single variable that can store multiple values.
Think of it as a set of drawers (or sequence of elements)

- The drawers as a whole, have a label, the List name
- Each individual drawer is numbered, starting at 0 (zero)
    - These are called the ***List indexes***

Lists are a type of **collection** or **container.** There are other types that we will visit in later portions of this course.

Declare a List in Python using square brackets [ ]

```
myList = [1, 2, 3, 4, 5] # A List with 5 values
```

You can also declare an empty List. You may find that you are not sure about what you're going to be storing in your List.

```
myList = []
```

Accessing List Elements: Recap

We can use individual List elements like any other variable:

```
userNames = ["Joe", "Jane", "Jack"]
print(f"User is called {userNames[1]}")

--Output--
What is the output?
```

```
grades = [72.0, 56.5, 61.1]

// Add up the grades and divide by 3 to get the average
averageGrade = (grades[0] + grades[1] + grades[2]) / 3 # Why parenthesis?
print(averageGrade)

--Output--
63.2
```

‘for’ Loops and Lists

Python Lists are well-suited for "for" loops. We can use the Python `len()` function to discern the size of the list:

```
userNames = ["Josh", "Ollie", "Ben"]

for i in range( len(userNames) ):
    print(userNames[i][i])
```

The first index `[i]` is accessing the userName element of index 0. The second index `[i]` is referring to the characters within the Strings contained within the element

# What is outputted onto the screen when the value of i is equal to one?
# What is the value of `len( userNames )` based on the List declaration?
# When writing a for loop we must provide values for the step, stop, and start. What are the default values of each?
# Why must we provide the for loop with one value?

Input and Output

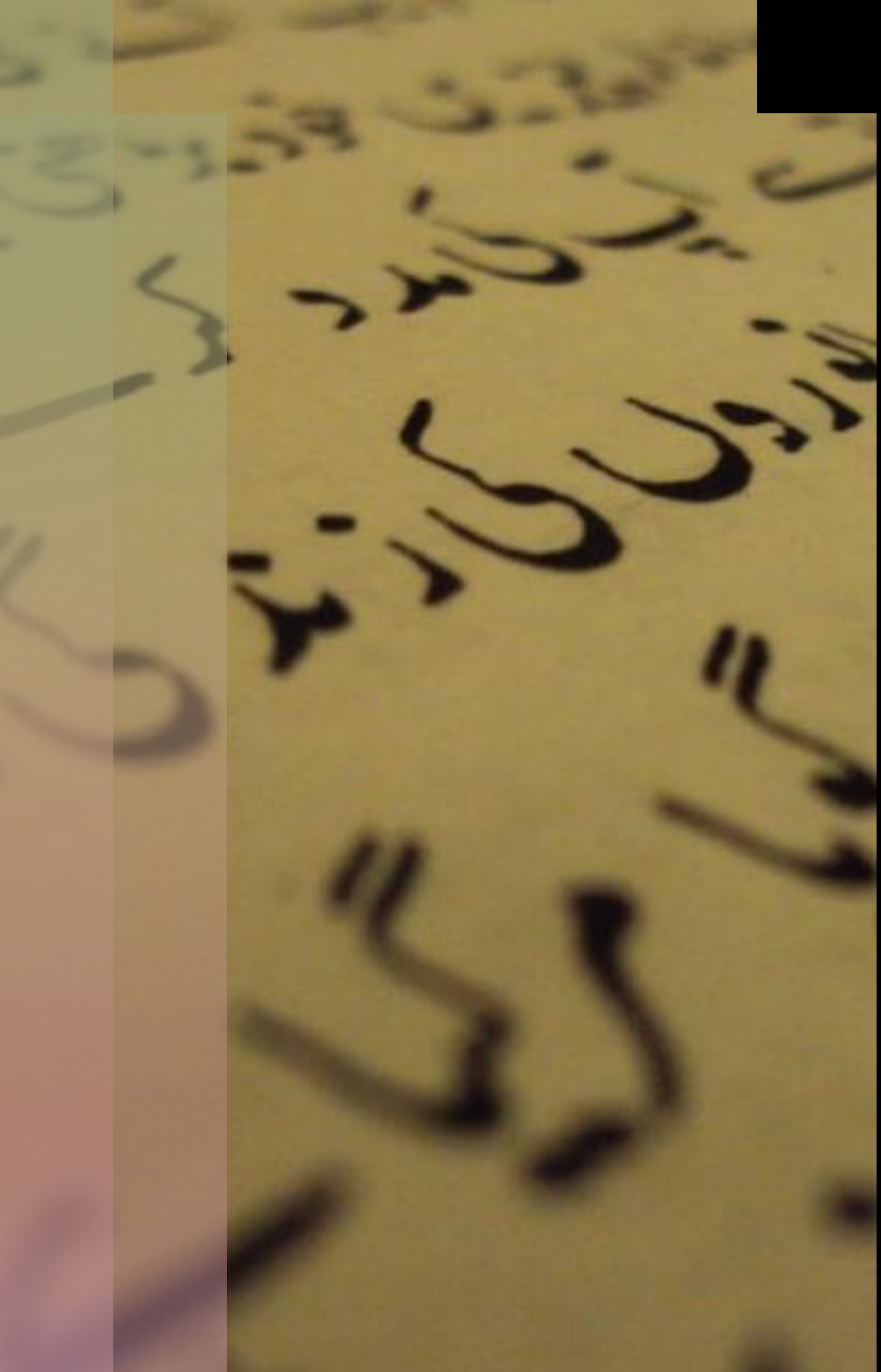
Computer programs need to interact with the outside world to do anything useful

Some examples include:

- Command line environments
- Files
- Networks
- Graphical user interfaces

In Python, we use the `input()` function to prompt users for an input, and the `print()` function as a simple way to output information to the console

Files can be written in text or binary. For this module we will focus on text outputs and general operations such as Open, Read/Write, and Close



Writing to a Text File

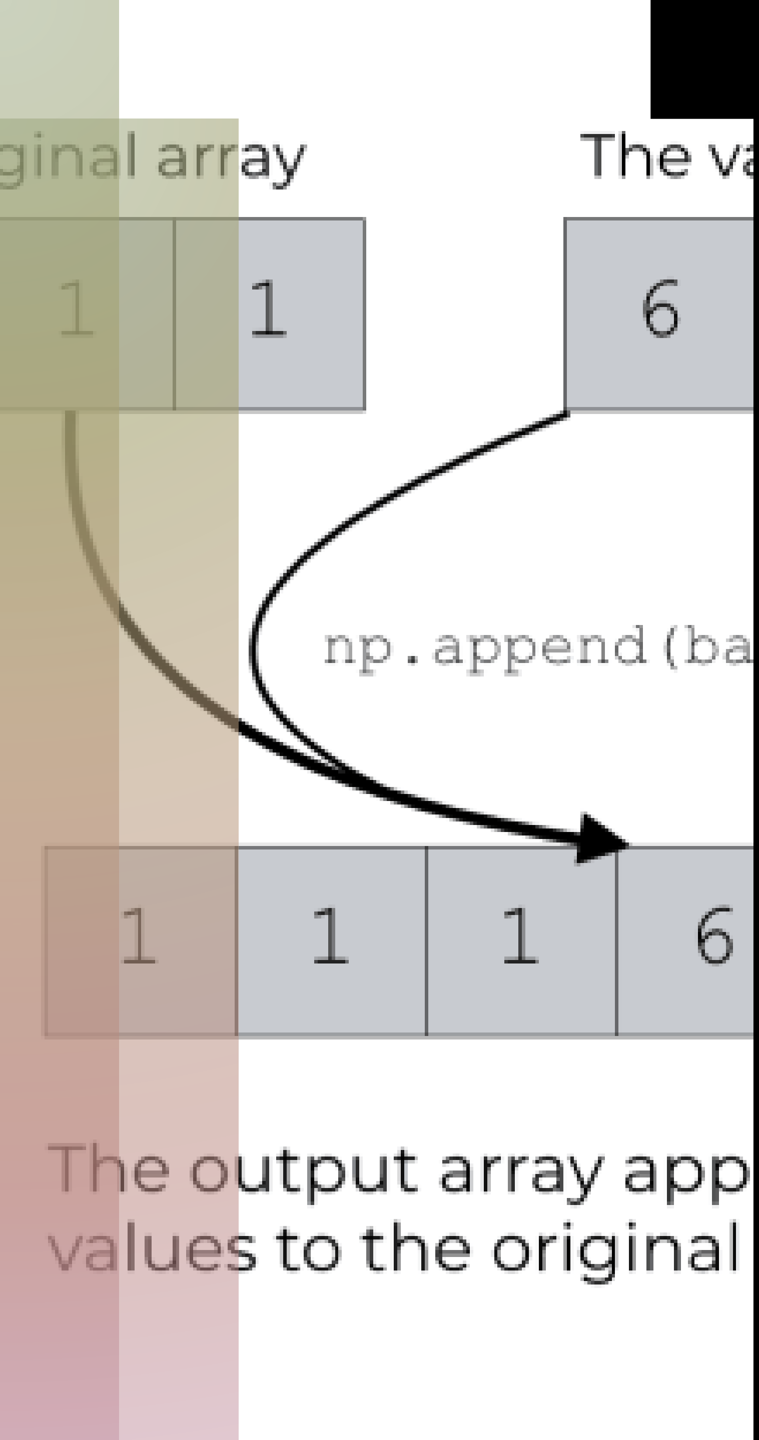
We can use the `open()` function to create and/or open a text file for writing and then use the `write()` function to write text to it

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

Note that the `with` statement has its own scope like the `if`, `for`, and `while` statements. The code within the `with` block will be executed. Once the block is exited the file will be automatically closed.

If you re-run this code, you will find that the previous text file contents have been overwritten.

So, how do we get around this...



Write File Modes

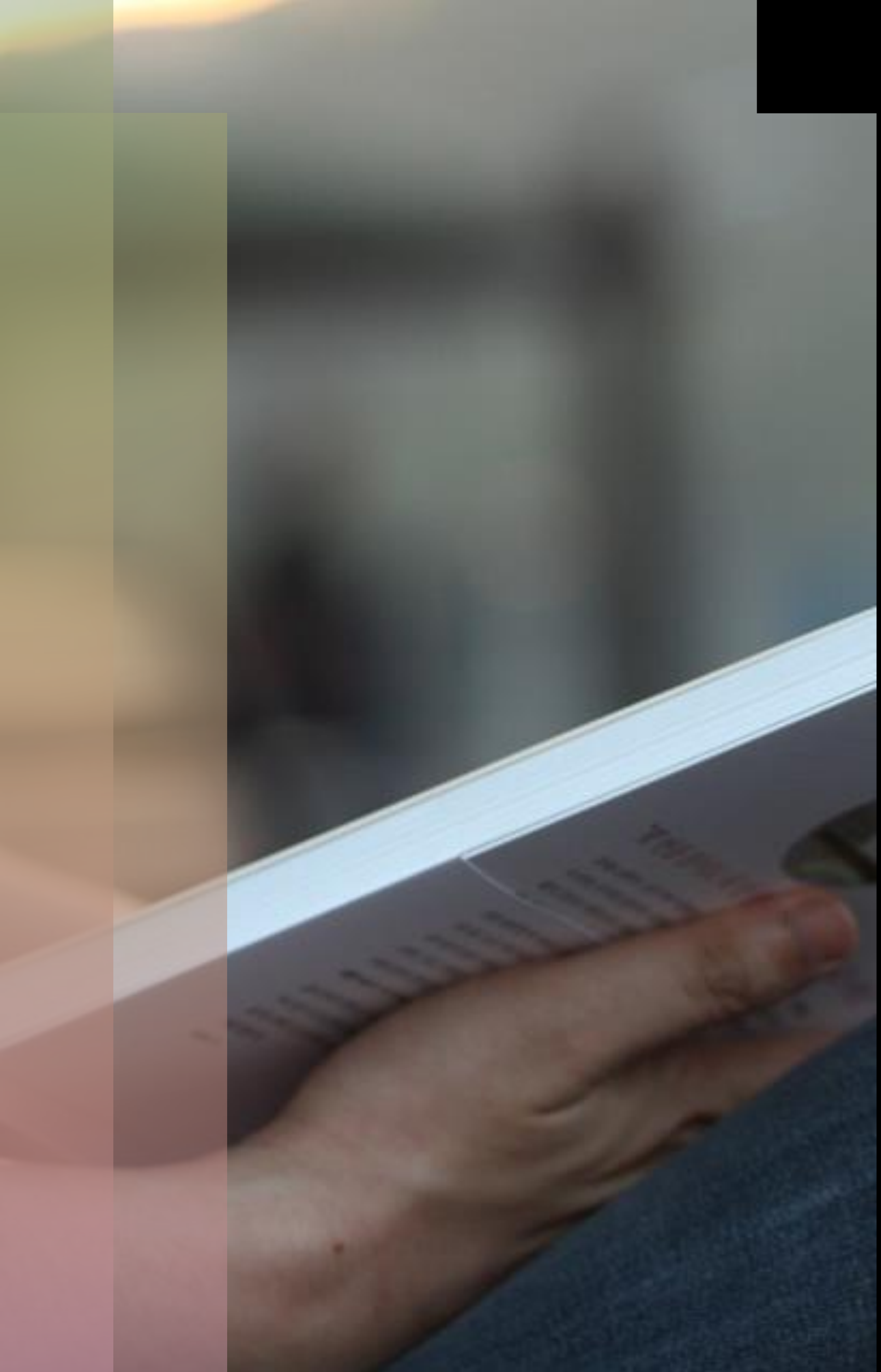
The open() function provides many modes for writing. We just covered the write "w" mode which creates and/or rewrites over the contents of a file

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 1

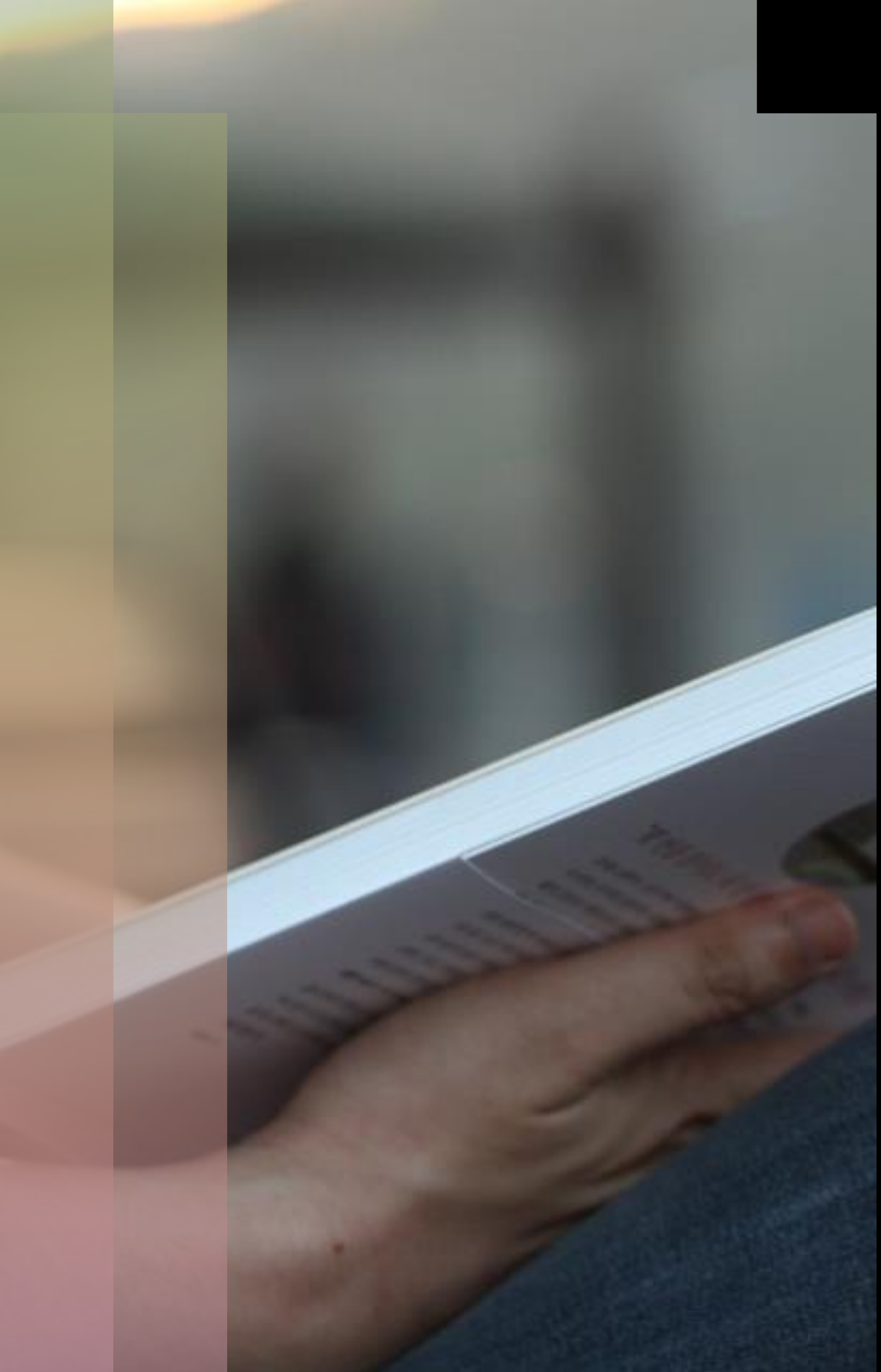
We can also use the `open()` function to create and/or open a text file for reading and then use the `read()` function to read the entire contents from a file

```
# Open a file for reading
with open( "test.txt", "r" ) as file:
    contents = file.read()
    print(contents)
```

In this context the entirety of the `"test.txt"` file is read into a string variable called `contents`. This string variables is then outputted onto the console

The default variable type for a `file.read()` is a string

How do we convert this to other types?



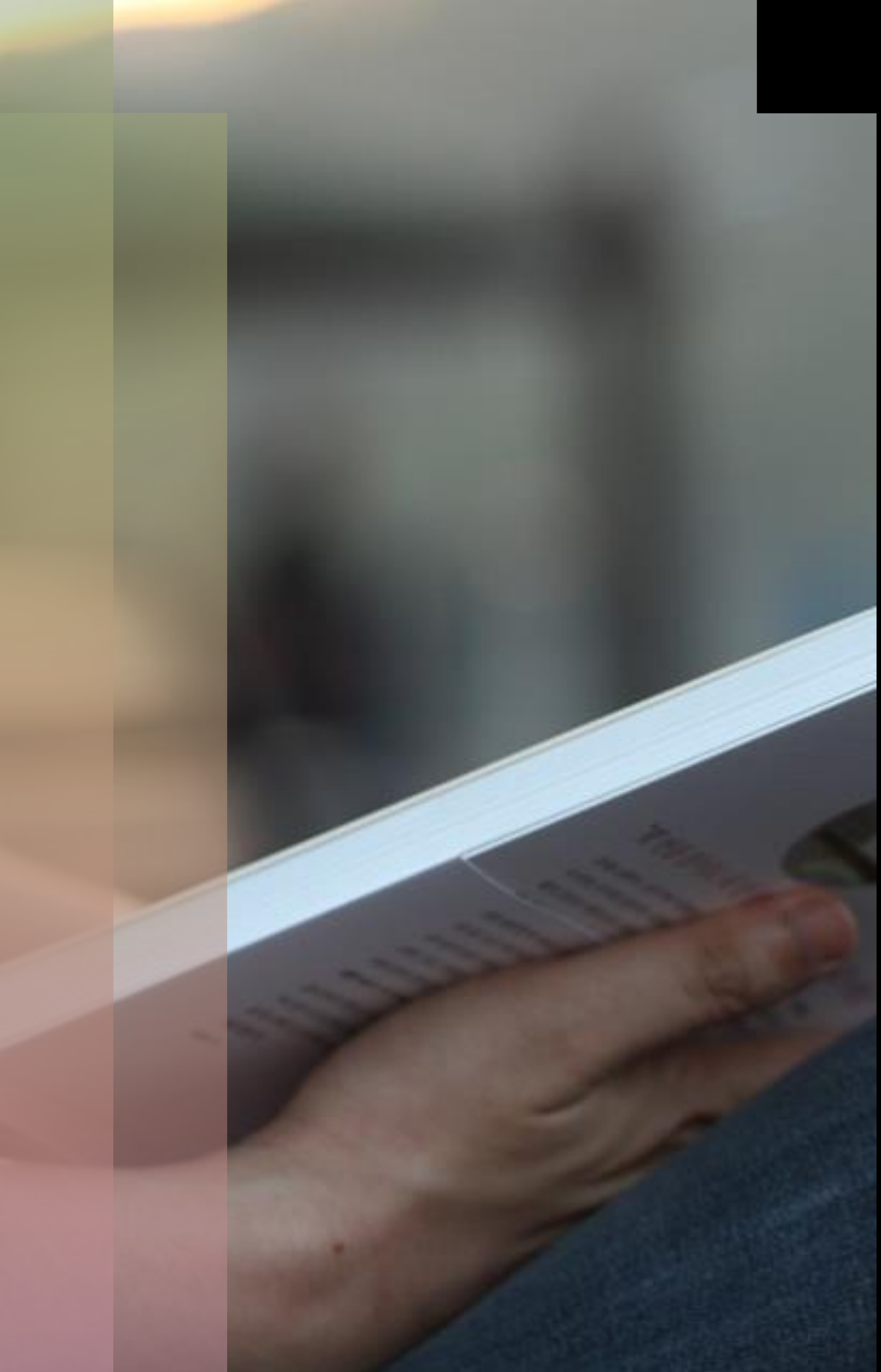
Reading a Text File 2

We can also use the `readline()` function to read the contents of the file line by line

```
# Open a file for reading
with open( "test.txt", "r" ) as file:
    line = file.readline() # Reads line 1
    while (line): # loop if line string is !empty
        print(line)
        line = file.readline() # Reads line 2
```

In this context the individual lines of the `"test.txt"` file are assigned to the `line` variable, and are then printed onto the console

The `while` loop will read lines from the `"test.txt"` file so long as the `line` variable is not an empty string (i.e., ""). An empty string is ***Falsely*** will always evaluate to `False` in Python, providing us with a perfect escape from the loop



Reading a Text File 3

We can also use the `readlines()` function to read the contents of the file line by line, into a Python List

```
# Open a file for reading
with open( "test.txt", "r" ) as file:
    lines = file.readlines() # Read into List
    for line in lines:
        print(line)
```

Let's say that the file contained three names Ben, Ollie, and Josh. The resultant List will be as following:

```
lines = ["Ben", "Ollie", "Josh"]
```

Using a for loop we can iterate over the lines List and print out the names

The individual elements of lines can be used too i.e., `lines[0]`

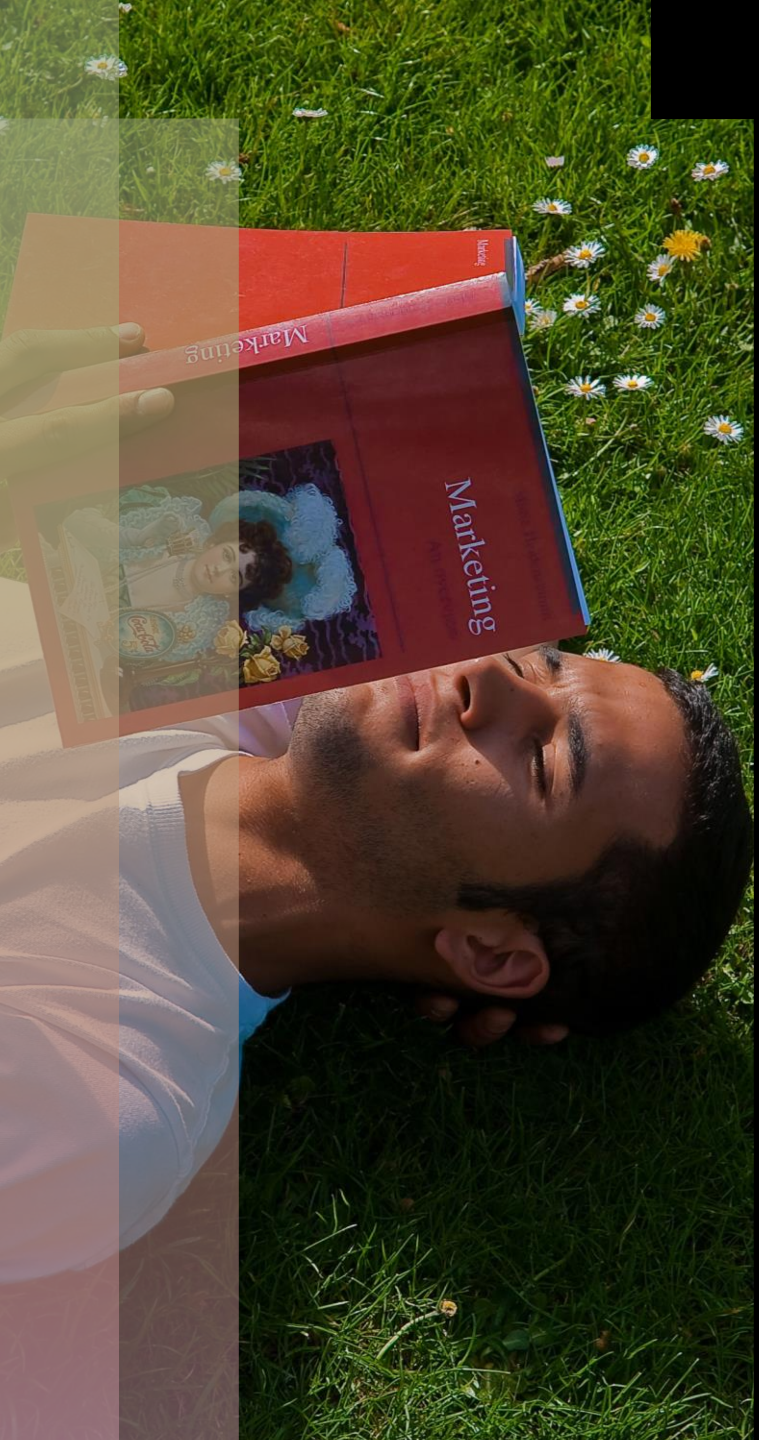
Side Note

Can anyone see the difference between the for loop (below) and those we have encountered in earlier sessions?

```
# Open a file for reading
with open( "test.txt", "r" ) as file:
    lines = file.readlines() # Read into List
    for line in lines:
        print(line)
```

We **do not** need to use the `range(start, end, step)` function when iterating over the elements of a List (or any other inerrable/collection) for that matter

The `for` loop iterates over the **elements** of the `lines` List and **assigns** the **current element** to the `line` variable. The current element is printed out in the example above



Read File Modes

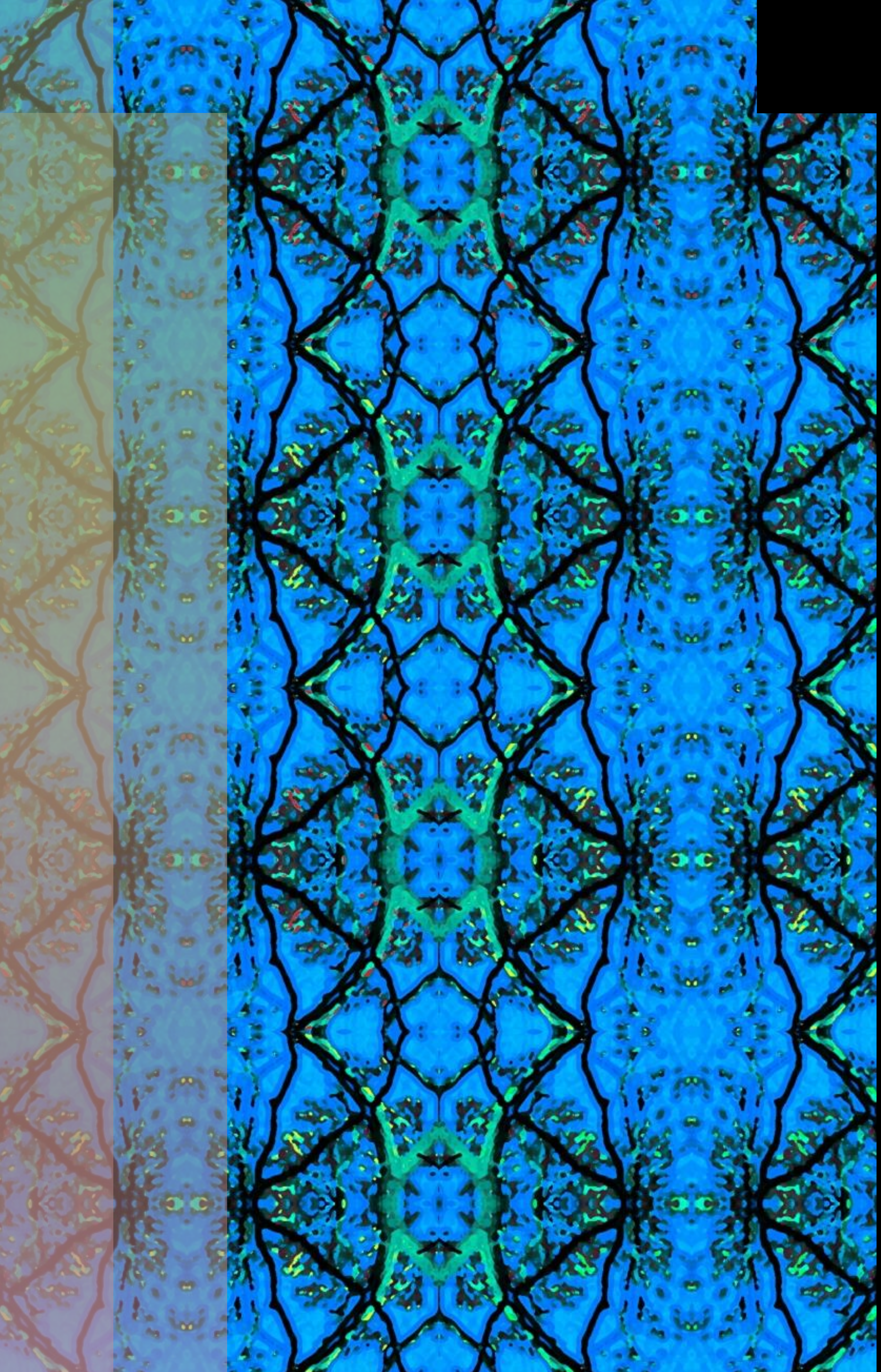
The open() function provides many modes for reading. We just covered the read "r" mode which reads the contents of an existing file

There is also a read text "rt" and a read binary "rb" mode

Read text "rt" is the exact same as read "r" but is our way of explicitly specifying text mode (can be useful to limit codes usage)

```
# Open a file for writing (creates it if non-existing)
with open( "test.bin", "rb" ) as file:
    contents = read()
    for byte in contents:
        print(hex(byte)) # printing the hexadecimal
        representation of each byte
```

We will not be using binary files in this module, but it doesn't hurt to know that you can do this



Program Length / Repetition

Most of the programs we have seen and written have been short
- However, real world programs are much longer
- Millions of lines of code are not unusual

You should start to notice pieces of code that are repeated in different parts of the program:

```
#... Lab Exercise 1
print("Press Enter to continue...")
input()

#... Lab Exercise 2
print("Press Enter to continue...")
input()
```

- Is there some way to reduce this repetition?

Reusable Code

We have used loops to repeat code many times

- Loops repeat the same code over and over
- On the last slide there is some code that is repeated, but also some code that isn't repeated (the exercises), so a loop isn't appropriate

Instead of thinking about this specific program, think generally:

- These two lines are frequently reused:

```
print("Press Enter to continue...")
input()
```

- The code is self-contained
- It performs a clearly identifiable task: wait for user to press enter

It would be useful to have a quick way to execute this code anytime we want…

Functions

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

Here’s an example of a function **definition**:

```
def WaitForEnter():
    print("Press Enter to continue...", end="")
    input()
```

And examples of a function calls:

```
#... Lab Exercise 1
WaitForEnter()
#... Lab Exercise 2
WaitForEnter()
```

Overall Program Structure

```python
def PrintName(name):
    print(f"Hi... {name}")

def main():
    # Define a name
    firstName = "Umar"
    sirname = "Arif"

    # Concatenate strings
    name = firstName + " " + sirname

    # Print the name to the console
    PrintName(name)

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

This is how a function is put into your overall program

Functions do not go inside of the main function
- Main is itself a function

They go at the same level as Main either before or afterwards

In this code main is always executed first
- Other functions are only executed if they are called
- Main acts as the **entryway** into the code and an easy way for us to order our statements and logic

Reasons to Use Functions

Why use functions?

- Clearly, they save typing
- The code becomes more readable
  - Shorter, simpler code is easier to read
  - Note how the name `PrintName` makes the meaning of the code clear
- The code is less prone to bugs
  - Writing the same code many times – several chances at writing a bug
  - With just one copy of the code - only one chance at writing a bug
    - And easy to find the bug if we do so
- The code is more flexible
  - If you want to make a change to the `PrintName` display, for example display the sirname before the firstName…
  - …then only change the code in one place, which will affect every call

Form of a Function Definition

A closer look at a function ***definition***:

```
    [A]def [B]WaitForEnter([C]):
         [D]
```

[A] The keyword def is used to define a function in Python

[B] This is a name we choose for the function. The style guide calls for PascalCase when declaring functions

[D] These brackets are required. We can send data to the function in here if we wish, otherwise we leave the brackets empty

[D] The function code goes here.

Calling a Functions

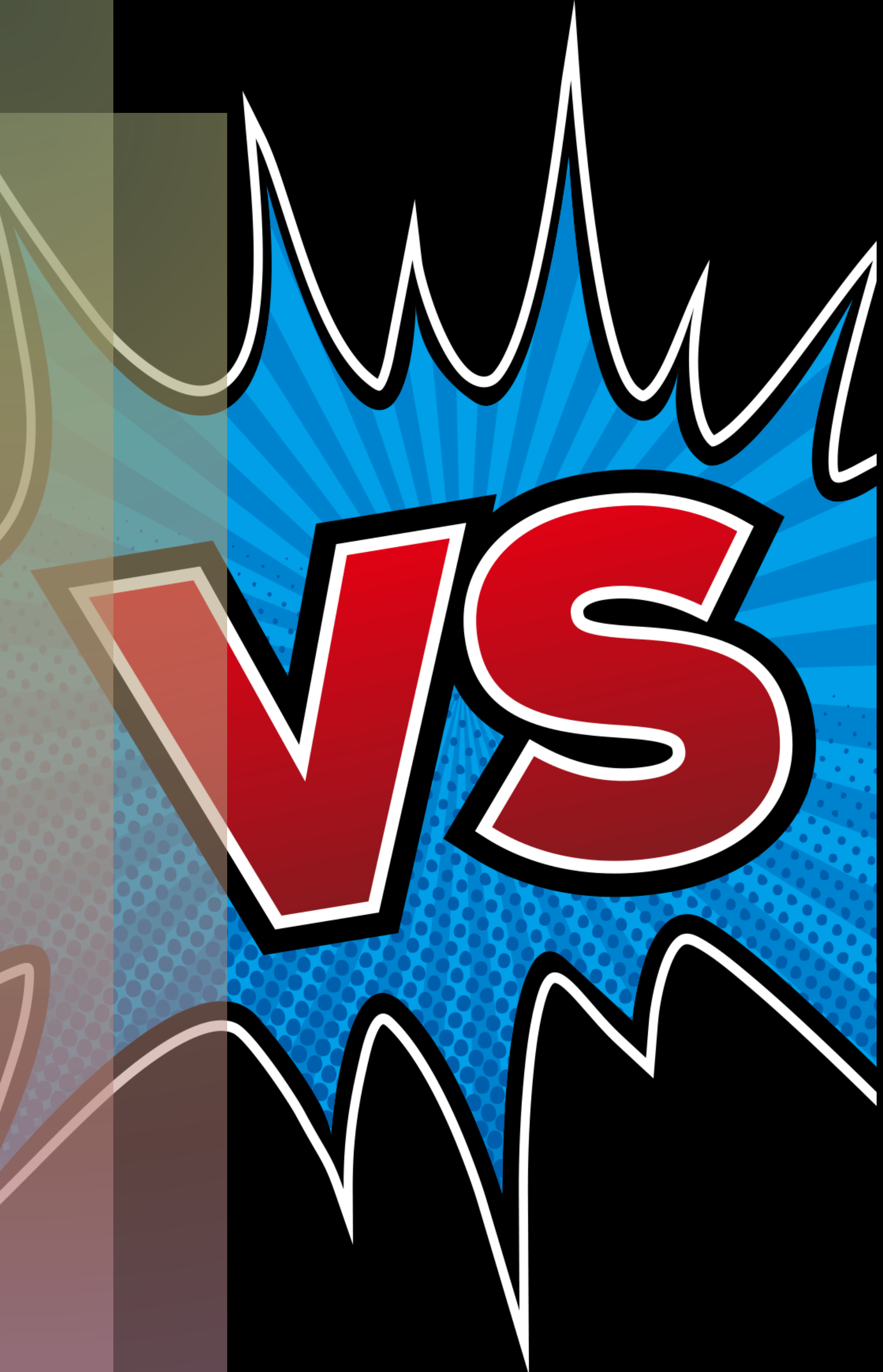
Given this function definition:

```
def WaitForEnter():
    print("Press Enter to continue... ", end="")
    input()
```

We call the function like this:

```
# Code before call
WaitForEnter()
# Code after call
```

- You don’t write anything apart from the function name when calling a function
- When the function call is reached, the function code is executed
- After the function is complete, it ***returns*** to the code after the call



Method vs Function?

The discussion so far has mostly mentioned functions, but occasionally the word method has been used. Why?

- Python is (just about) an *object-oriented* programming language – it uses concepts called classes and object. We will cover these at some later point.
- In a class we use the term method for this topic
- Outside of a class we use the term function

- Unlike languages like C# where everything is a class, Python is a language where everything you write in not encapsulated in a class, so we use both methods and functions

- However, a method and a function are very similar concepts.
- People tend to use the words interchangeably, so I need to mention both
  - I would like everyone to use the correct terms if possible.

Returning a Result from a Function

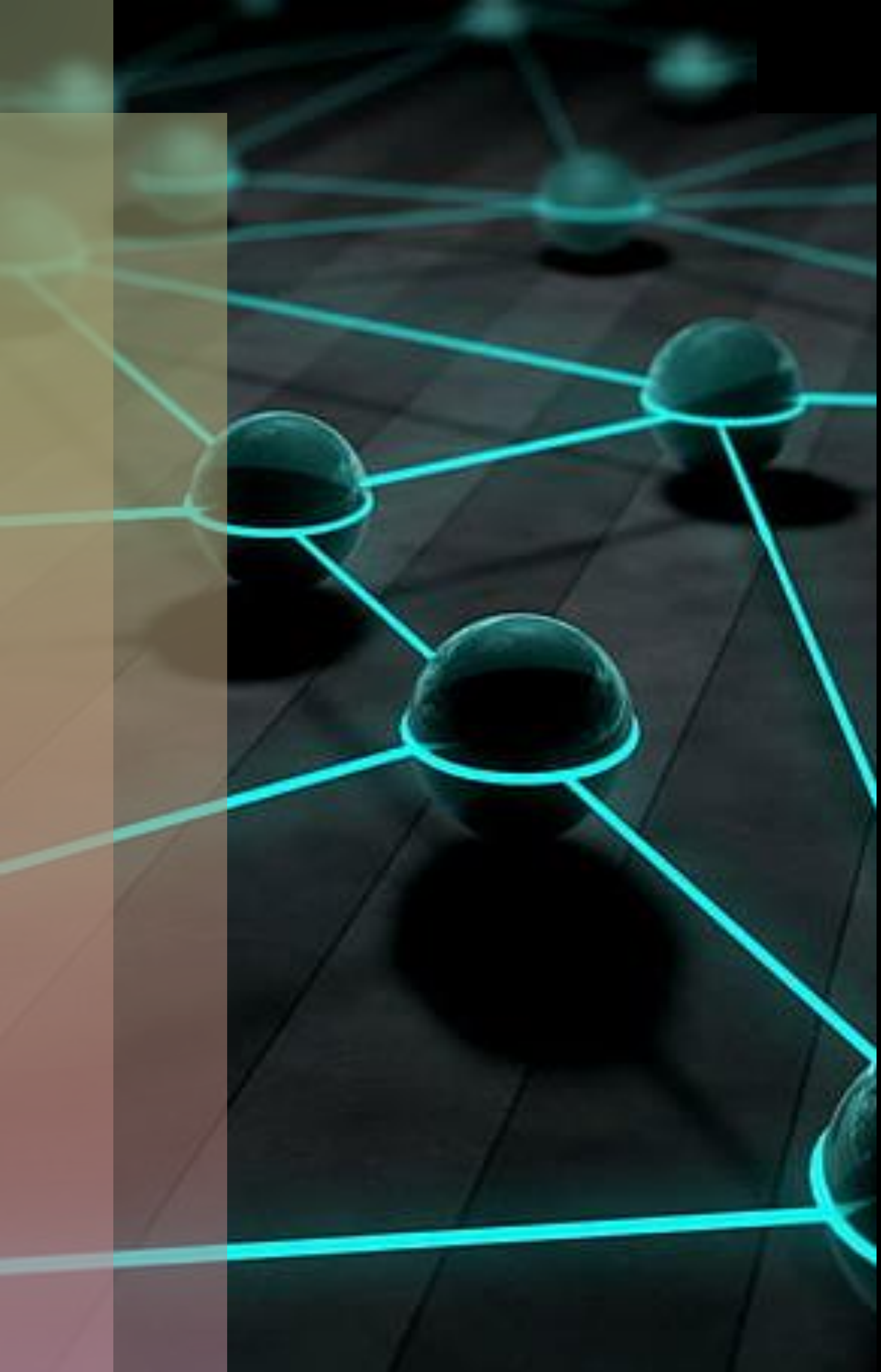
A function can return a result

- It can perform a task or a calculation that gives an "answer"

The returned result must be a single value of a given type

- We do not indicate which type the function returns in the function definition – Python can figure it out on its own
- Add a `return` statement in the function code

```
# This function returns a float - this can be
assigned to a variable
def ReadFloatFromUser(message):
    userFloat = float( input(message))
    return userFloat
```



Using a Returned Result

Function calls that return a value can be used in a different way

- We can use the function call as though it was a value:

```
# This function returns a float – this can be
assigned to a variable
def ReadFloatFromUser(message):
    userFloat = float(input(message))
    return userFloat

message = "What is your weight in kg?"
weightInKG = ReadFloatFromUser(message)

message = "What is your height in cm?"
heightInInches = ReadFloatFromUser(message)
```

We can put this code anywhere we can put a float as this function returns a float

Parameters: Send Data

We can pass values into a function to help it perform its task

- These values are called parameters or arguments
- Correct word depends on the context, often used interchangeably
- We can pass one or more values
- The passed values initialise variables available inside the function
- Specify parameters in the function definition:

```
# Initial value for 'year' passed from caller
def AgeFromYear(year):
    # Use year as a variable in the function
    approxAge = 2023 - year
    print(f"Age is {approxAge}")

# Pass an integer to initialise the parameter
year in the function
AgeFromYear(1996)
```

Parameters are Variables

Consider this example again:

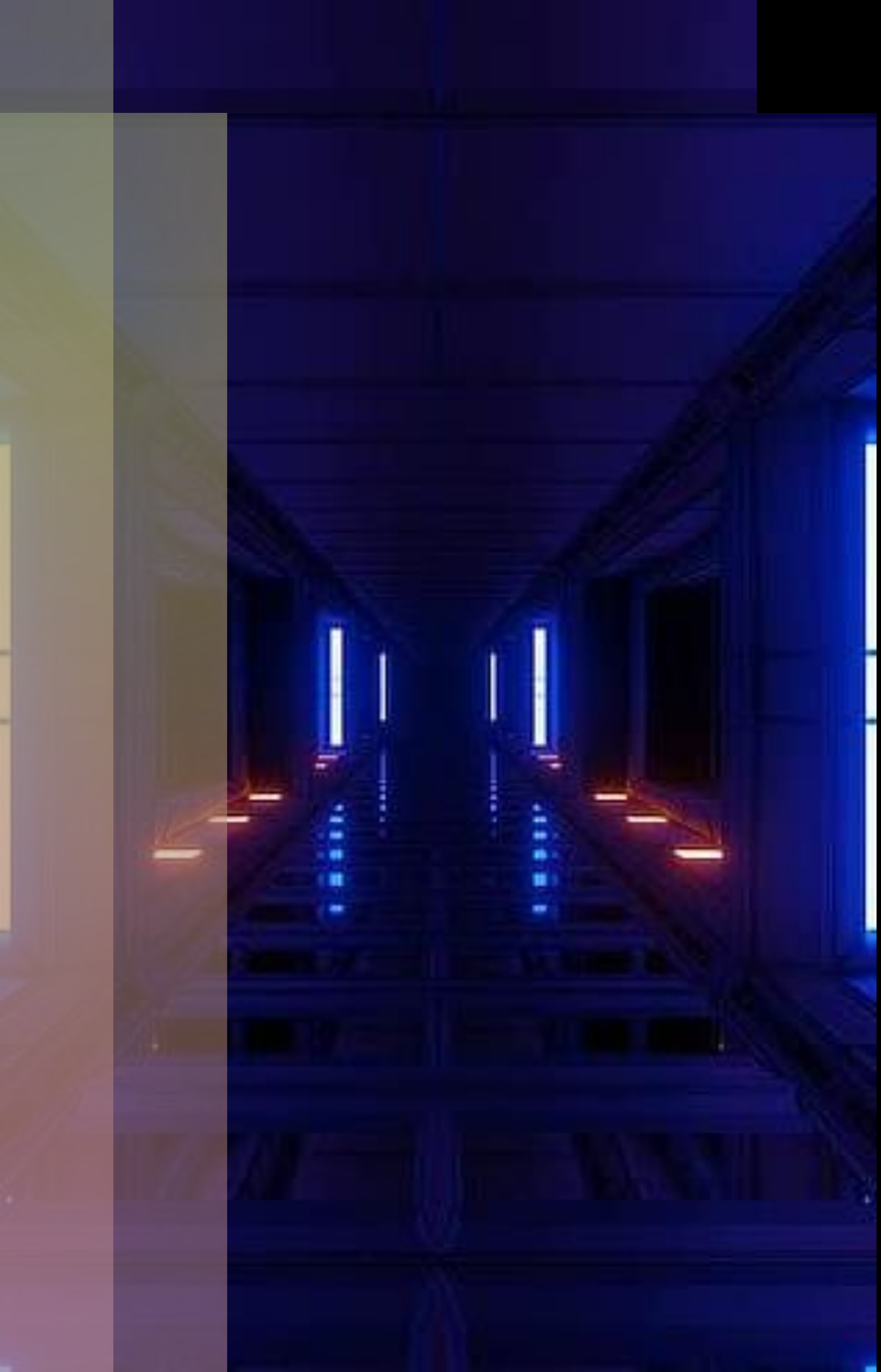
```
    def AgeFromYear( year ):
        approxAge = 2017 - year
        print(f"Age is {approxAge}")
```

```
userAge = AgeFromYear( 1980 )
```

By passing 1980 to the function, it behaves like this:

```
    def AgeFromYear():
        year = 1980
        approxAge = 2017 - year
        print(f"Age is {approxAge}")
```

year is a variable inside the function block, initialised to 1980 here



Using Parameters and Returning a Result

Most useful functions accept parameter(s) and return a result:

```
    CelsiusToFarhenheit( celsius ):
        farhenheit = celsius * 13.0 / 9.0 + 32.0
        return farhenheit # Sends value back to main

-----------------------------------------------------

tempInC = float(input("Enter a temp in Celsius: "))
tempInF = CelsiusToFarhenheit( tempInC )
print(f"That temperature in Farhenheit is {tempInF}")
```

- In this case we are passing a value from the user to the function
- ...the user variable `tempInC` *initialises* the parameter `celcius`
- The names don't match **AND** they don't need to
- Collect the function result in `tempInF` and display it

Common Mistake: Calling Functions

Functions are a little complex when you first see them
A common mistake is to remember how to write the function definition correctly, but to forget how to call the function

- This is a common mistake:

```
def ReadFloatFromUser(message):
    userFloat = float(input(message))
    return userFloat

# ...some code to try and use the function
userNumber = def ReadNumberFromUser(message)
```

Don't just copy the function definitions to call a function

Correct style:

```
weightInKG = ReadFloatFromUser("Weight in KG: ")
```

Common Mistake: Ignoring Return Value

A frequent novice mistake is to ignore the return value:

```
message = "What is your weight in kg? "
# Bug: return value is ignored and so is lost
ReadNumberFromUser(message)
# What to write here?
print("Your weight is {?????}")
```

The correct call should be something like:

```
weight = ReadNumberFromUser(message)
```

If you don’t use the result from the call in some way, it is lost

Although this does not appear as an error and appears to be sensible, it could be the sign of a mistake.