TechTalent Academy
Safeguarding Policy

*"Protecting an adult's right to live in **safety, free from abuse and neglect**. It is about people and organisations working together to **prevent and stop both the risks and experience of abuse or neglect**, while at the same time making sure that the **adult's wellbeing is promoted** including, where appropriate, having regard to their views, wishes, feelings and beliefs in deciding on any action. This must recognise that adults sometimes have complex interpersonal relationships and may be ambivalent, unclear or unrealistic about their personal circumstances."*

If you have a safeguarding concern, please raise this with your tutor or via the safeguarding link on our website:

https://www.techtalent.co.uk/safeguarding-statement

TechTalent's safeguarding lead is: **Max Ruddock**

Starter Activity.

Code it:

Ask the user's age. If they are 18 or over, display the message "You can vote", if they are aged 17, display the message "You can learn to drive", if they are 16, display the message "You can buy a lottery ticket", if they are under 16, display the message "You can go Trick-or-Treating".

Starter Activity.

Answer

```
age = int (input("What is your age? "))
if age >=18:
    print("You can vote")
elif age == 17:
    print ("You can learn to drive")
elif age == 16:
    print ("You can buy a lottery ticket")
else:
    print("You can go trick-treating")
```

TechTalent Academy

Data Science Course

Python Fundamentals Part 3

Lesson Objectives.

- For Loops
- Nested Loops
- While Loops
- Functions and Procedures

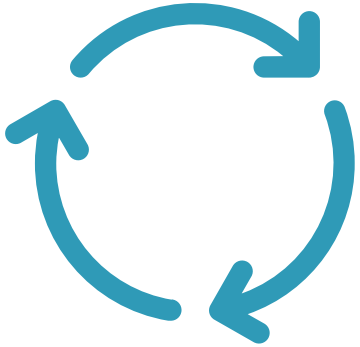
Iteration.

When you run your program, this will usually run from the first line of your code to the last line of your code.

A **loop** allows you to run through a particular section of your code until a specific condition is met.

For example, if your program allows a user to enter a password, you will not want to exit the program straight away after someone gets their password wrong once.

This could often be as a result of simple human error. Most programs recording passwords will give the user a set amount of goes to try and enter their password correctly. This is done using a loop.



For loops.

Lists can be iterated over to retrieve each value, and apply the same code multiple times to each value in the list.

```
farm_cows = ["Winnie the Moo", "Dasiy", "Milkshake", "Buttercup"]
for cow in farm_cows:
    print(cow, "says moo!")
```

✓ 2.4s

```
Winnie the Moo says moo!
Dasiy says moo!
Milkshake says moo!
Buttercup says moo!
```

For loops structure.

Syntax:

index: 0 1 2 3

list= ["Caroline", "Lea", "Mike", "Ismael"]

```
for items in list:
    print("items")
```

indentation needed

output

```
Caroline
Lea
Mike
Ismael
```

For loops with indexes.

For loops can also be used with an incrementing index. This is where a value gets added to each time.

Using the range function returns a list of integers:

- range(end_index) - list from 0 up to, but not including, this value
- range(start_index, end_index) - list from start_index up to end_index
- range(start_index, end_index, step) - list from start_index up to end_index, with specified incrementation

```
range(4)          # [0, 1, 2, 3]
range(10, 19)     # [10, 11, 12, 13, 14, 15, 16, 17, 18]
range(10, 19, 2)  # [10, 12, 14, 16, 18]
range(18, 9, -2)  # [18, 16, 14, 12, 10]
```

For loops with indexes.

To print out every other cow from farm_cow, indexes can be used with the range function.

```
# For loops with indexes
farm_cows = ["Winnie the Moo", "Dasiy", "Milkshake", "Buttercup"
for i in range(1, 4, 2):     # range(1, 4, going up in 2s
    print(farm_cows[i], "says moo!")
```

✓ 0.3s

```
Dasiy says moo!
Buttercup says moo!
```

Transfer and control - continue.

Continue is used within a loop to instantly go to the next iteration

```python
# continue
farm_cows = ["Winnie the Moo", "Dasiy", "Milkshake", "Buttercup"]
for cow in farm_cows:
    if cow == "Milkshake":
        continue
    print(cow, "says moo!")
```

✓ 0.4s

```
Winnie the Moo says moo!
Dasiy says moo!
Buttercup says moo!
```

Transfer and control - pass.

Pass is used as a placeholder for code to come. An if statement body can't be empty so therefore the placeholder "pass" is used. It means to pass over this.

```
# pass
farm_cows = ["Winnie the Moo", "Dasiy", "Milkshake", "Buttercup"]
for cow in farm_cows:
    if cow == "Milkshake":
        pass
    print(cow, "says moo!")
```

✓ 0.4s

```
Winnie the Moo says moo!
Dasiy says moo!
Milkshake says moo!
Buttercup says moo!
```

Transfer and control - break.

Break is used to stop the execution of a loop.

```python
# break
farm_cows = ["Winnie the Moo", "Dasiy", "Milkshake", "Buttercup"]
for cow in farm_cows:
    if cow == "Milkshake":
        break
    print(cow, "says moo!")
```

✓ 0.5s

```
Winnie the Moo says moo!
Dasiy says moo!
```

Task: For loop.

Ask the user to enter their name and then display their name three times.

Nested loops.

Sometimes need and possible to put a loop inside a loop. The "inner loop" will be executed one time for each iteration of the "outer loop“.

Be careful it can get complicated – Consistent indentation helps.

```python
# Nested loop
for i in range(3):
    for j in range(4):
        print(i,"*",j,"=",i*j)
```

✓ 0.4s

```
0 * 0 = 0
0 * 1 = 0
0 * 2 = 0
0 * 3 = 0
1 * 0 = 0
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
2 * 0 = 0
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
```

```python
# Nest loop example
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

✓ 0.5s

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Nested loops.

```
for i in range(3):
    for j in range(4):
        print(i, j)
```

i	0	1	2
j	0 1 2 3	0 1 2 3	0 1 2 3

The second loop **j** will act on the first loop **i**

Nested loops.

```
for i in range(3):
    for j in range(4):
        print(i*j)
```

i	0	1	2
j	0*0	0*1	0*2
	1*0	1*1	1*2
	2*0	2*1	2*2
	3*0	3*1	3*2

The second loop **j** will act on the first loop **i** by multiplying each item of **i** (0, 1, 2) by its own items **j** (0, 1, 2 ,3)

While Loops.

While loops will loop over code as long as a condition is true. The condition is checked after the loop code has completed.

The code below does the following:

- i has the value of 0
- 1 is being added onto the value of i each loop for the code whilst the value of i is less than or equal to 12

```
i = 0

while i <= 12:
    print("The value of i is:", i)
    i += 2 # Short hand for i = i + 1
```

✓ 0.4s

```
The value of i is: 0
The value of i is: 2
The value of i is: 4
The value of i is: 6
The value of i is: 8
The value of i is: 10
The value of i is: 12
```

Task: While loop.

Set the total to 0 to start with. While the total is 50 or less, ask the user to input a number. Add that number to the total and print the message “The total is... [total]”. Stop the loop when the total is over 50.

For loops vs While loops.

For loops and while loops can be coded to do the same thing:

```
for i in range(3):
    print(i)
```

✓ 0.3s

```
0
1
2
```

```
i = 0
while(i < 3):
    print(i)
    i = i + 1
```

✓ 0.5s

```
0
1
2
```

However:

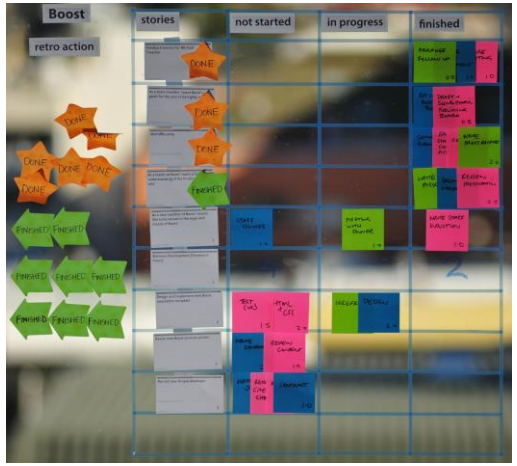
- **For loops should be used when it is known ahead of time the number of iterations**
- **While loops should be used when it is not known ahead of time the number of iterations**

What are functions?

- A function is a **reusable** block of code that is used to perform a specific action
- Functions are mostly a part of a larger piece of code that solved the problem
- Using functions will make life easier in terms of programming

How to functions help us?

Code Modularisation



Program Organisation

Increased manageability

How do you define functions and procedures?

The **def** method is used in Python for this**!**

- Any input arguments need to be placed **inside** the brackets while defining the function
- The code block within every function:
  1. Starts with a colon (:)
  2. Follows indentation

Functions and Procedures.

- **Procedures output a result straight away, but there will be times where you want a procedure to return some data back to you**
  - For this, we use a **function**

- **Functions must have an extra line of code within them to create the return variable, which passes the return value back. For example:**
  - **return answer**

- **We can do many things with the return value, including:**
  - Store it in a variable
  - Use it in a calculation
  - Or even use it as a parameter in another procedure/function…!

Procedures.

Syntax:

```
def hello_procedure(name):
        print("Hello "+name)

hello_procedure("Mohamed")
```

Defining the procedure with the keyword def

Indentation needed

Procedure calling

Don't forget the colon

Parameter(s)
Can be empty!

Argument(s)
Can be empty!

Functions.

Syntax:

```
def hello_function(name):
        return "Hello "+name
a= hello_function("Mohamed")
print(a)
```

Defining the function with the keyword def

Indentation needed
return keyword

Function calling

Don't forget the colon

Parameter(s)
Can be empty!

Argument(s)
Can be empty!

Function Examples.

- **We can do many things with the return value, including:**
  - Store it in a variable
  - Use it in a calculation
  - Or even use it as a parameter in another procedure/function…!

```
# Function
def number(number1, number2):
    '''
    Function to sum up numbers
    with 2 parameters
    '''
    answer = number1 + number2
    return answer

added_number = number (5,5)

print(added_number + 20)

✓ 0.1s

30
```

Multiline comments with triple quotes: **docstrings**

Function Examples.

- Something great about Python functions is that you can use the return value of one function as the parameter of another function! Consider the following:

- Follow the rules of BODMAS to work out the answer

```python
def addition(number1, number2):
    answer = number1 + number2
    return answer

def subtraction(number1, number2):
    answer = number1 - number2
    return answer

def multiply(number1, number2):
    answer = number1 * number2
    return answer

print(multiply(addition(2,2), subtraction(10,8)))
```

✓ 0.1s

```
8
```

Task: Sub program.

Define a subprogram that will ask the user to enter a number and save it as the variable “num”. Define another subprogram that will use “num” and count from 1 to that number.

Parameter Passing.

Sometimes we need to pass values to the function/procedure.

We need to declare these within the () when creating the function/procedure

Important!

- The order in which the parameters are declared is important for the order the values are passed to them.
- You can then use these parameters within the function.
- Allows for code reusability
- DRY (Don’t Repeat Yourself!)

Parameter passing Examples.

```python
# Passing in parameters
def sayhi(name):
    print("Hello " + name)

sayhi("Chester")
sayhi("Sam")
```

✓ 0.8s

```
Hello Chester
Hello Sam
```

```python
# Passing in multiple parameters
def sayhi(name, age):
    print("Hello " + name + ", you are " + age)

sayhi("Chester", "24")
```

✓ 0.5s

```
Hello Chester, you are 24
```

```python
def procedure_2(inp_name,inp_age):  # think as placeholders
    print("Hello", inp_name + "! You will be ", str(inp_age + 1), "For your next birthday!")

name = input("What is your name? ") # goes into placeholder 1
age  = int(input("What is your age? ")) # goes into placeholder 2

procedure_2(name, age)
```

✓ 9.4s

```
Hello Georgina! You will be  32 For your next birthday!
```

Parameter passing.

There are different types of arguments that can be passed to parameters:

-Positional arguments: order and number matters

-Keyword arguments: number matters, not the order

-Arbitrary arguments: *args and **kwargs are used to bypass the numbers of arguments

Bypassing the number of keyword arguments.

Arguments used either a * or ** in front of a parameter name:

*args acts as a tuple:

```
def function (*args):
    print(args)

function(1,23,42)
✓ 0.3s
```

```
(1, 23, 42)
```

** kwargs acts a dictionary:

```
def function (**kwargs):
    print(kwargs)

function(a=1,b=23,c=42)
✓ 0.2s
```

```
{'a': 1, 'b': 23, 'c': 42}
```

Only for keyword arguments

Lambda Function.

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax:

lambda parameter(s): expression

```
multiplication= lambda x,y: x*y
multiplication(6,10)
```

✓ 0.5s

```
60
```

techtalent academy

Class & Object.

self needs to be the first parameter (can be named differently)

```
class Sum():

    def __init__(self, num1, num2, num3):
        self.num= num1+num2+num3

    def sum(self):
        print(self.num)

sum_calculation= Sum(1,23,34)
✓ 0.2s
```

output

```
sum_calculation.sum()
✓ 0.2s
58
```

1. Define the class name (use capital first letter and a self-describing name)

2. We need to create a self-defining function called __init__ (special keyword recognised by python)

3. We need to create a function that we will execute our desired code

4. We need to create an object that will associate our arguments with the class parameters

We can perform the sum of our numbers using the sum function in our class. Class calling: “instantiation”

Python module.

- A python module is a python file with the extension **.py** such as: **module1.py**
- The file will contain all code that will be run one time and executed by the terminal shell
- Make sure to be with the terminal shell in the working folder where is present the module1.py file

  so python can access it
- Modules can be imported with the keyword: **import** module1
- Functions within modules can be accessed using: **module1.function_name**

Plenary.

Print `i` as long as `i` is less than 6.

```
i = 1
      i < 6
  print(i)
  i += 1
```

In the loop, when the item value is "banana", jump directly to the next item.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    
  print(x)
```

Create a function named `my_function`.

```
                    :
  print("Hello from a function")
```