Functions in Python

Functions are re-useable blocks of code. To **invoke** a function, you "**call**" it by placing a set of parenthesis "()" after the function name. A simple example of a function might be a message to contact tech support.

```
# function - A block of reuseable code
#            place () after function name to invoke it

def contactTechSupport():
    print("----------------------------------------")
    print("If you have any problems, please contact")
    print("Technical Support on 03 9876 1234")
    print("----------------------------------------")
    print()

print("Welcome to the program!")
print()
contactTechSupport()
```

Positional Arguments

You can send data to a function using **positional arguments** so the function can use this data to achieve some task and return a result.

A **positional argument** means the arguments passed must be in the **same order** as the parameters defined by the function.

```
def contactSupport(department, phone):
    print("----------------------------------------")
    print("If you have any problems, please contact")
    print(f"{department} on {phone}")
    print("----------------------------------------")
    print()

print("Welcome to the program!")
print()
contactSupport("Accounts", "03 9876 1234") # <- order matters
```

After sending **arguments** to a function, there may be instances where you want a result returned. This is achieved using the **return** statement.

```
def add(x, y):
    z = x + y
    return z

def subtract(x, y):
    z = x - y
    return z

def multipy(x, y):
    z = x * y
    return z

def divide(x, y):
    z = x / y
    return z

print(add(1, 2)) # <-- add function becomes the returned value
print(subtract(1, 10))
print(multipy(2, 20))
print(divide(15, 3))
```

Outputs:

```
3
-9
40
5.0
```

Default Arguments

If a function requires arguments but the arguments have been omitted, then the function can be set up to use **default arguments** and IF arguments ARE sent, they will over-ride the defaults.

```
def contactSupport(department='support', phone='9876 1289'):
    print("----------------------------------------")
    print("If you have any problems, please contact")
    print(f"{department} on {phone}")
    print("----------------------------------------")
    print()

print("Welcome to the program!")
print()
#contactSupport("Accounts", "03 9876 1234") # <- order matters
contactSupport()
```

Another Example:

Write a **function** that requires **arguments** for price, discount and GST. Set default argument for discount to 0% and GST to 10%. Default arguments must always come AFTER required arguments

The solution without default arguments:

```
def net_price(sale_price, discount, gst):
    return sale_price * (1-discount/100) * (1+gst/100)

print(net_price(500,50, 10))
```

If we set the default discount to 0% and the GST to always be 10%:

```
def net_price(sale_price, discount=0, gst=10):
    return sale_price * (1-discount/100) * (1+gst/100)

print(net_price(500))
```

The **argument** position is important. You could call the function as follows:

print(net_price(500))

print(net_price(500, 20))

print(net_price(500, 20, 15))

Exercise:

We will create a “count up” module.

```
def count(start, end):
    for x in range(start, end+1): # <-- 2nd arg is exclusive
        print(x, end=' ')
        time.sleep(1) #<-- time modules sleep method pauses 1s
    print("DONE!")

count (0, 10)
```

If most counting will start at 0, we could set start=0

```
def count(start=0, end):
```

this will fail because “parameter without a default follows parameter with a default.

```
def count(end, start=0): #<-- parameter without default first
    for x in range(start, end+1): #<-- 2nd arg is exclusive
        print(x, end=' ')
        time.sleep(1) #<-- time modules sleep method pauses 1s
    print("DONE!")

count (5) # <-- this will be the “end” parameter
```

Keyword Arguments

An optional feature when sending arguments to a function is that we can prefix an argument with the name of the parameter. Keyword arguments can help with code readability.

These are **keyword arguments**. Note that the first argument has no keyword, this will be treated as a positional argument.

```
def hello(greeting, title, first, last):
    print(f"{greeting} {title} {first} {last}")

hello("Hello", title="Mr.", last="Down", first="Bob")
```

Order does **not** matter when using keyword arguments. Keyword arguments, however, **must follow** positional argument if mixing them.

```
def hello(greeting, title, first, last):
    print(f"{greeting} {title} {first} {last}")

hello(title="Mr.", greeting="Hello", last="Down", first="Bob")
```

Built in Arguments

The “print” function has some in-built arguments, including **end** and **sep**:

```
for number in range(1, 11):
    print(number, end=" ")   #<-- "end" is a keyword argument
                             #     within print
```

```
print("1", "2", "3", "4", "5", sep="~")
# "sep" is another keyword argument within print
```

Exercise:

Create a function to generate a phone number. We need to pass in a country code, area code, first 4 digits then last 4 digits.

```
def get_phone(country, area, first, last):
    return f"+{country} {area} {first} {last} "

phone_num = get_phone(country=61,area=3,first=9876,last=7890)

print(phone_num)
```