GOVERNMENT OF WESTERN AUSTRALIA

TAFE WA South Metropolitan

Functions

(Session 14)

Overview

- Introduction to Functions
- Writing a Function
- Defining Function Parameters
- Arguments
- Default Values
- Returning Values

Introduction to functions

What is function?

- A block of code (group of statements) that runs when it is called.
- It avoids repetition and makes code more organized

- Where are functions found?
  - **Default functions** built into Python (print(), input(), etc.). See https://docs.python.org/3/library/functions.html
  - **Pre-installed modules** that are used less often, but still built-in (e.g., random).
  - **Created by you**! You can create your own functions in your code and use them as you will.
  - Also, **classes**, but that is beyond the scope of this unit.

Introduction to Functions - Examples

```
import random   # imports Python random module
# randrange() returns a random number between the given range
print(random.randrange(2, 5))
# sample() returns a given sample of sequence
my_list = [1, 2, 3, 4, 5, 6, 7]
print(random.sample(my_list, k=2))  # any 2 of the items

# built in functions such as abs() and pow()
print(abs(-6))    # returns the absolute number
print(pow(10, 2))  # returns base to the exponent
```

```
2
[7, 1]
6
100
```

Writing a functions

- Function names have the same naming rules as variables.
- A function is defined using the **def** keyword
- Syntax:

  def *functionName(parameter):*

  functionBody

- *Example:*

```
def my_function():
  print("Writing a function is easy")
```

Activity 1: Writing a functions

- Let’s give it a go!

```
def my_function():
    print("Enter an integer amount")

print("Starting")
var1= int(input())
var2= int(input())
print("Ending")
```

- What happened? Why?

Activity 2: Writing a functions

- We didn't invoke it (call it) at all, so it never ran our function code.
- Let's give it another go!

```
def my_function():
    print("Enter an integer amount: ", end="")

print("Starting")
my_function()     # call it
var1= int(input())
my_function()     # call it again
var2= int(input())
print("Ending")
```

- What happened?

Writing a functions

- There are two important rules with using functions!

1. **You cannot call a function which isn't already known at the runtime of its call.**

- That is, you can't write the call (**my_function()**) before you define the function. It hasn't loaded into memory yet.

```
my_function()

def my_function():
    print("Enter an integer amount")

print("Starting")
var1= int(input())
```

- This will error (NameError: name not defined). Much like variables, you have to define the function before calling on it.

Writing a functions

- There are two important rules with using functions!

2. **You can't have a function and a variable use the same name.**

```
def my_function():
    print("Enter an integer amount")

print("Starting")
my_function= int(input())
```

- *To an extent this still runs, **BUT**, **my_function** now only holds the value of the input, it has completely forgotten about our function.*

Parameters vs Arguments

- The parameter is a variable that is handled by the function in a few different ways.
  - This variable ONLY exists inside the function that we defined, and it can only be placed between the brackets.
  - The value of the variable is assigned WHEN you call the function.
- One thing to note:
  - Parameters only exist inside a function. **def funct_name(***parameters***)**
  - Arguments exist outside of the functions. These carry the value that is passed to the parameters. **funct_name**(*arguments*)

Parameterised Functions

- Let's look at our previous example and add a parameter:

```
def my_function(number):
    print("Enter an integer amount: ", end="")

my_function()    # missing positional argument
```

- This will error (TypeError: missing 1 required position argument)
- You'll notice the **number** parameter. It is ONLY usable inside our my_function() function.
- But currently it's not going to do anything different, let's try a few tests to get an idea.

Activity 3: Parameterised Functions

- Let's try another test, we'll add 7 as an **argument** to the function call:

```
def my_function(number):
    print("Enter an integer amount: ", end="")

my_function(7)    #  positional argument
```

- What do we get? Why do you think it happened?

Defining function parameters

- You can have as many parameters in your function as you like, but the more you have, the more difficult it is to remember them all on the call.
- Let's create another parameter!

```
def my_function(item,number):
    print("Enter an item", item, "\nEnter an integer amount", number)
```

Defining function parameters

- Now, let's feed the two parameters with two arguments.

```
def my_function(item,number):
    print("Enter an item: ", item, "\nEnter an integer amount: ", number)

my_function("Milk",7)
my_function("Bread",1)
```

- What do we get?

```
Enter an item:  Milk
Enter an integer amount:  7
Enter an item:  Bread
Enter an integer amount:  1
```

Arguments

- If we need to switch them around, we can do that easily, but the output will be different.

```
def my_function(item,number):
    print("Enter an item: ", item, "\nEnter an integer amount: ", number)

my_function(7, "Milk")
my_function("Bread",1)
```

```
Enter an item:  7
Enter an integer amount:  Milk
Enter an item:  Bread
Enter an integer amount:  1
```

- What happened?

Arguments – keyword argument passing

- We can also assign values regardless of the position by knowing the parameter name, this is called keyword argument passing.


```
def my_function(item,number):
    print("Enter an item: ", item, "\nEnter an integer amount: ", number)

my_function(number = 7, item = "Milk")
my_function(item = "Bread", number = 1)
```

```
Enter an item:  Milk
Enter an integer amount:  7
Enter an item:  Bread
Enter an integer amount:  1
```

- What happened now?
- You need to make sure you use the exact parameter name, otherwise it will error.

Arguments

- Let's look at a function:

```
def add_numbers(x,y,z):
    print("Sum is:", x + y + z)

add_numbers(3,5,7)
```

- We're only using positional arguments.

```
Sum is: 15
```

Arguments

- You can mix both positional and keyword arguments BUT, you must put positional arguments before keyword arguments.

```
def add_numbers(x,y,z):
    print("Sum is:", x + y + z)

add_numbers(3,y=5,z=7)
```

```
Sum is: 15
```

- We have the argument of 3 for the parameter passed as a **positional** argument.
- We have the arguments b and c passed as **keyword** arguments.
- What happened?

Arguments

- You can mix both positional and keyword arguments BUT, you must put positional arguments before keyword arguments.
- Let's mix them:

```
def add_numbers(x,y,z):
    print("Sum is:", x + y + z)

add_numbers(5,x=5,z=7)
```

```
Traceback (most recent call last):
  File "<string>", line 4, in <module>
TypeError: add_numbers() got multiple values for
    argument 'x'
```

- Now what happened? Why?

Default Values

- Parameters can be set to have a default value.

```
def add_numbers(x=3,y=5,z=7):
    print("Sum is:", x + y + z)

add_numbers()
```

```
Sum is: 15
```

- *What do we get now?*

Default Values

- Or even...

```
def add_numbers(x,y=5,z=7):
    print("Sum is:", x + y + z)

add_numbers(3)
```

- *What do we get now?*

```
Sum is: 15
```

Returning values

- So far we have had functions that have an effect. They print some text for us. But what about if we are calculating something in a function.
- Much like an equation it will have a result.
- For this we need: return

Returning values

- There are two ways to use this:
  1. return
     - With just the keyword and nothing after it, it will cause the function to finish immediately in place, returning to the point it was called without further running the rest of the function. This automatically happens at the end of the function.
  2. return *expression*
     - With an expression after the return it will still immediately finish and return from the function, but it will return the expression value as the result.

Activity 4: Returning Value


```
def multiply_numbers(x,y):
    return x * y


z = multiply_numbers(2, 3)
print(z)
```

```
6
```

Activity 5: Returning values


```
def divide_numbers(x,y):
    if y == 0:
        return    # just the keyword and nothing after
    else:
        return x/y

x = float(input("Enter number 1: "))
y = float(input("Enter number 2: "))
result = divide_numbers(x, y)
if result:
    print(result)
else:
    print("Division by zero is illegal")
```

```
Enter number 1: 3
Enter number 2: 4
0.75
```

```
Enter number 1: 3
Enter number 2: 0
Division by zero is illegal
```

Questions?