Functions & Scripting.

Functions:

Functions: are a way that allow you to encapsulate a task.

Encapsulations: is a way of carrying out a whole series of steps with one simple command.

Defining Functions:

```
def cylinder_volume(height, radius):
    pi = 3.14159
    return height * pi * radius ** 2
```

That was an example of function definition.

As seen the function header is made of multiple components:

- keyword : (def).
- Function name : (cylinder_volume) (it follows the same naming convention of variables).
- Function Argument : (height, radius) (must be in parentheses).

The function body contains:

- The code written to do a task, here you can refer to arguments variables and define new variables within the scope of the function.
- A return statement followed by an argument which is used to output a value from the function.

Examples :

```
def readable_timedelta(days):
    # use integer division to get the number of weeks
    weeks = days // 7
    # use % to get the number of days that remain
    remainder = days % 7
    return "{} week(s) and {} day(s).".format(weeks, remainder)
```

```
def population_density(population, land_area):
    return population/land_area
```

Function Scopes:

Variable Scope: Refers to which part of your code a variable can be referenced, or used from.

It's important to consider scope when using variables in functions. If a variable is created inside a function, it can only be used within that function. Accessing it outside that function is not possible.

EX:

```
# This will result in an error
def some_function():
    word = "hello"

print(word)
```

here the variable word is only local to the function some_function.

An upside to this is you can have multiple variables with the same name with each local to their own functions.

Global scope variables: variables which are defined before a function which means it can be accessed before said function.

EX:

```
# This works fine
word = "hello"

def some_function():
    print(word)

some_function()
```

Notice that we can still access the value of the global variable `word` within this function. However, the value of a global variable can not be **modified** inside the function. If you want to modify that variable's value inside this function, it should be passed in as an argument .

Lambda Expressions:

lambda expressions are useful for creating quick functions that aren't needed later in code.

They are quite useful for higher order functions or functions that take other functions as arguments.

EX:

```
#Normal Function
def multiply(x, y):
```

```
    return x * y
```

made into:

```
#lambda function
multiply = lambda x, y: x * y
```

Lambda Function components:

- keyword lambda: indicates that this is a lambda function.
- one or more arguments separated by (,) followed by (:) (x, y)
- an expression that is evaluated and returned (x * y)

Useful Terms:

Function: A block of code that has a name, but doesn't run until we tell it to.

Function Call: A statement that makes a function run.

Argument: A value that we can pass to a function when we call that function.

Method: A function associated with an object.

Scripting Notes.

Notes:

- function eval: when taking an input from user as a math expression python can interpret it and solve it on it's own.

  EX:

```
result = eval(input("Enter an expression: "))
print(result)

INPUT: 3 * 6
OUTPUT: 18
```

Handling Errors:

Types of Errors:

- **Syntax errors:** occur when Python can't interpret our code, since we didn't follow the correct syntax for Python.They're likely caused by a Typo.
- **Exceptions :** occur when unexpected things happen during execution of a program, even if the code is syntactically correct.

Handling Errors:

We can use `try` statements to handle exceptions.

Structure:

```
try:
    # some code
except ValueError:
    # some code

or

try:
    # some code
except (ValueError, KeyboardInterrupt):
    # some code
```

you can access a specific error message as follows:

```
try:
    # some code
except ZeroDivisionError as e:
   # some code
   print("ZeroDivisionError occurred: {}".format(e))

OUTPUT:
ZeroDivisionError occurred: integer division or modulo by zero
```

Reading and writing files:

Here's how we read and write files in Python:

- Reading a file:

```
f = open('my_path/my_file.txt', 'r')
file_data = f.read()
f.close()
```

or with

```
with open('my_path/my_file.txt', 'r') as f:
    file_data = f.read()
```

(With open is preferred more as it ensures the file is closed, freeing up memory)

Importing Files:

There are other variants of `import` statements that are useful in different situations:

- To import multiple individual objects from a module:

```
from module_name import first_object, second_object
```

- To import an object from a module and rename it:

```
from module_name import object_name as new_name
```

- If you really want to use all of the objects from a module, use the standard import module_name statement instead and access each of the objects with the dot notation.

```
import module_name
#the dot notation
module_name.object
```