**Chapter 7 - Custom Functions**

Custom functions are used for specific tasks.

* Define the function:

def function\_name(parameter1, parameter2,…):

Instruction

Instruction

Instruction

* One can either set no parameters, thus calling a function such as quote(), and expecting it to return simple prints(“bla bla bla’)
* Or one can insert parameters in the function such as
* def multiply(num1,num2):
  + Print(“the result of the multiplication is:’, num1\*num2)
* One can either have **optional** or **mandatory** parameters
* **Mandatory** parameters must be specified first. These are parameters that the user has to put into the function.
* **Optional** parameters are parameters that the user can choose to put into the function. If he chooses not to, a **default value** is put in for him.
* **E.g. def calculate(par1, par2, par3=5,par4=0):**

Result = (par1\*par2)/(par3 + par4)

Print(result)

* To pass an argument to par4 but not par3, we would use: **parameter\_name = value.**
* **Using keyword arguments, we can call the function specifying the arguments in the order we prefer, without respecting the positions of the definition.**

**Productive functions:**

* Until now we’ve seen voidfunctions– can’t be stored into a variable.
* By using the **return** statement, one can use the result of the function and assign it to a variable.
* E.g.

Def calculate(num1,num2,num3):

Result = num1 + num2 + num3

**return** result

>>> x = calculate(4,5,2)

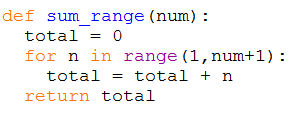
>>> x

10.0

. Global variables, placed outside a function, can be used within the whole program. **Local variables** on the other hand, are defined within the function and therefore cannot be called to use in IDLE shell.

. The docsting helps you understand what the function does ‘’’ ‘’’ when you call the function the string will appear.

**7.7 Functions with loops and conditional statements:**



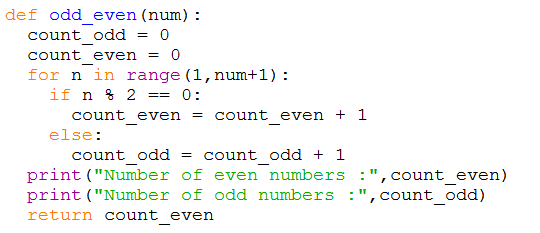
We must pass to the function an integer as an argument, which is later used in a for loop to calculate the sum of all the numbers between 1 and the number passed as an argument.

E.g. >>> sum\_numbers(8)

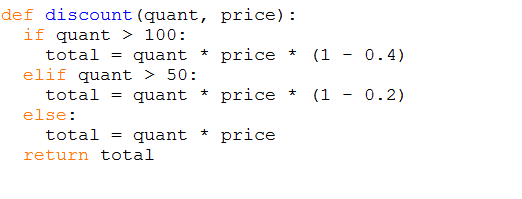
36

**The value returned by the function (36) is the sum of all the numbers between 1 and 8.**

**Other example:**



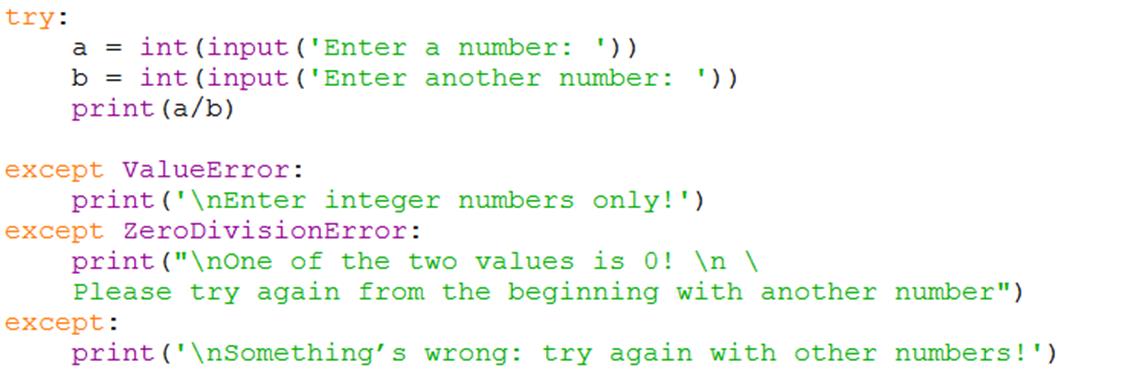
**Other example:**



**Exception handling:** **TRY and EXCEPT**

* An exception is an event triggered by an error. If we expect an error of some kind, we can write some code to handle it.
  + Syntax errors: error In how code is written
  + Runtime errors: error in code even if syntax is correct e.g. division by 0 or name not defined.
  + Semantic errors: when program is executed without producing error messages, but the results are not the correct ones – inconsistent or not expected – hard to find. i.e. python won’t show you these errors.

**Example:**

**Create a program that performs a simple division between two integer numbers entered by the user. The program must prevent from any kind of errors occurring while entering numbers:**

TypeError is also used for when non-numeric input is entered.