006: Functions

Learning Outcome:	Functions: in-built, custom.

Definitions/Concepts	
Functions	● Function is a *block of code*, which *runs* when it is *called*, we can *pass values* into it, and it can *return values or do something*. ● While writing many lines of code, there is always a chance that we need to code something which we have already coded. ● To reuse that part of code, we can convert it into a function and use it again just by writing the name of the function instead of all those lines of code.
Calling a function	● Calling a function means writing the function name to run it.
Parameters	● The values which can be passed in a function are called parameters or arguments of that function.
Types of functions	● Functions can be of two major types: ○ In-built functions: The functions like print(), int(), len(), range() etc are all inbuilt functions. ○ Custom functions: The functions that user can create are called custom functions.

Creating a function

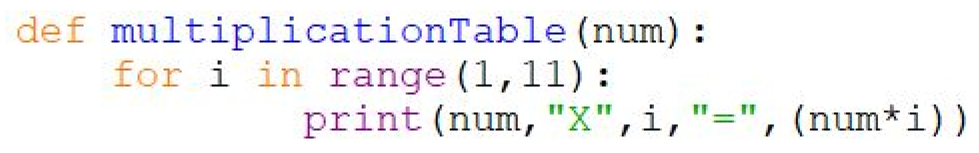
Syntax:

```
def functionName(parameters):
    statment1
    statement2
    return variable
```

- **def** is the keyword used in defining a function.
- The function name follows similar rules as of naming a variable:
    - Numbers and letters are allowed.
    - We can’t use spaces or special characters in it.
    - Keywords can’t be used as function names.
- Parameters/Arguments are the values which we pass to a function. It is optional, depending upon the nature of function.
- **return** is a keyword which gives the output of the function, if any.

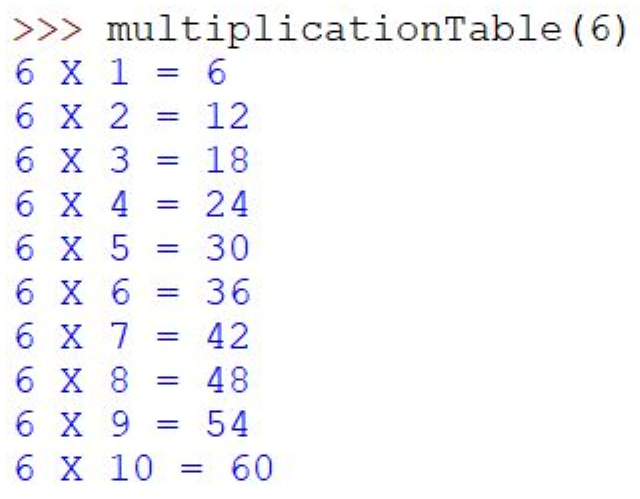
Example of creating and calling a function

To create a function to take a number and display its multiplication table.



```
def multiplicationTable(num):
    for i in range(1,11):
        print(num,"X",i,"=",(num*i))
```

Now to use the function, we can simply call it, as follows.

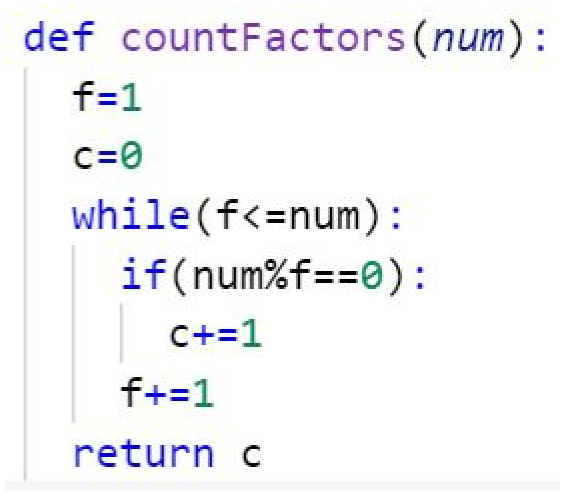


```
>>> multiplicationTable(6)
6 X 1 = 6
6 X 2 = 12
6 X 3 = 18
6 X 4 = 24
6 X 5 = 30
6 X 6 = 36
6 X 7 = 42
6 X 8 = 48
6 X 9 = 54
6 X 10 = 60
```

Activity links and Solutions

Student Activity 1: function

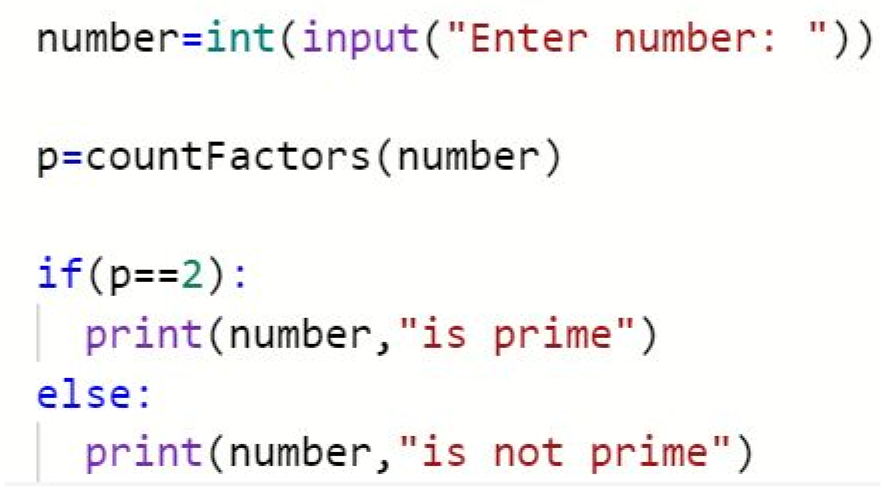
#Activity 1: Define a function to return the number of factors of a given number(including the number itself), for eg, factors of 6 are 1,2,3 and 6 itself, so no. of factors of 6 is 4



```python
def countFactors(num):
    f=1
    c=0
    while(f<=num):
        if(num%f==0):
            c+=1
        f+=1
    return c
```

- The variable **c** here acts as a counter, its value will increase by 1, when we get a factor.
- So, the no. of factors of **num** is stored in variable **c**
- The statement: **return c** will return the no. of factors

#Activity 2: Input a number from the user and using the above function check if the number is prime or not.



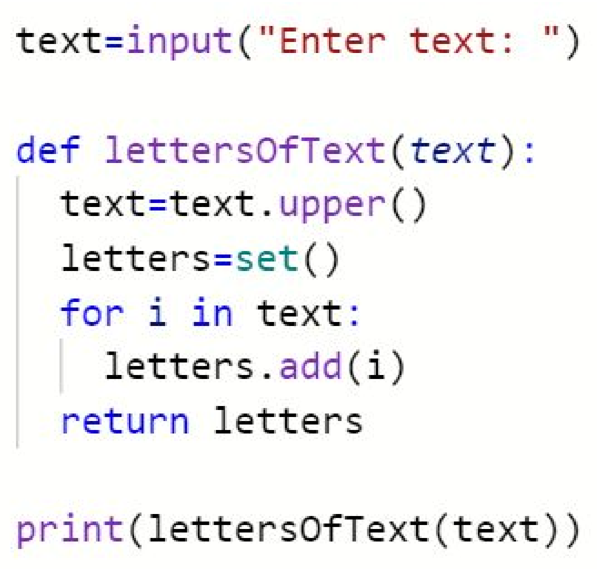
```python
number=int(input("Enter number: "))

p=countFactors(number)

if(p==2):
    print(number,"is prime")
else:
    print(number,"is not prime")
```

Function, **countFactors()** is called to find the no. of factors of **number**.

- Remember it contain **return** keyword, so on calling it, we get a value, so to store the value we are using another variable **p**
- We pass the variable **number** in the function.
- Now, **p** contains the no. of factors of **number**
- If the value of **p** is 2, then it is prime otherwise it is not prime, to check this we use the if conditional.

Student Activity 2: Letters in a Word

#Activity: Define a function to take a string as input and display all the distinct letters in the string in Upper case.



```
text=input("Enter text: ")

def lettersOfText(text):
    text=text.upper()
    letters=set()
    for i in text:
        letters.add(i)
    return letters

print(lettersOfText(text))
```

- The function **lettersOfText()** is taking parameter **text** of data type String and returning the variable **letters** of datatype set.
- .upper() functions turns text in upper case
- set() function creates an empty set in the variable letters
- .add() function adds one character from text in the set letters

Fun-fact

One can return multiple values in a function in Python.