Example of a void Function

The most simple example of a void function is function main, which is required in other languages but not in Python, which we have been using occasionally. Here is a short program with a very basic function main:

```
# This is a function definition.
def main():
    print("hi")


# Comment this out and see that nothing happens
# This is a function call.
main()
```

As previously discussed, we define our own functions with the keyword def, followed by the name of the function, followed by a pair of parentheses. Inside those parentheses there may be FORMAL parameters, which are variable names for the input values the function takes in. main has no formal parameters. Following the parentheses is the colon, to indicate that main is starting. What follows is the definition of the function main, indented inside of the def.

If you were to interpret the two line program (first two lines), nothing would happen. Defining a function doesn't execute anything. To execute anything you must CALL the function. This is what the last line does. A function call uses the name of the function and doesn't use the keyword def. It is followed by parentheses. Inside the parentheses are the ACTUAL parameters, of which main has none, since it has no formal parameters. This tells the computer to execute function main. Thus, this is the line that invokes the function, and then the function prints "hi".

More interesting example of a void function

Though many functions return values, it's not required for a function to return a value. Some functions may simply carry out a specified task and have no need to return any value to the function that called it. Consider the following function that prints out a given string a certain number of times:

```
def printString(myStr, n):
    print(myStr*n, end="")
```

We can use this function to print out a simple design:

```
def main():

    printString(' ', 2)
    printString('*', 1)
    printString('\n',1)
    printString(' ', 1)
```

```
    printString('*', 3)
    printString('\n',1)
    printString('*', 5)
    printString('\n',1)
    printString(' ', 1)
    printString('*', 3)
    printString('\n',1)
    printString(' ', 2)
    printString('*', 1)
    printString('\n',1)

main()
```

Note: It's clear that for this particular example, the use of the function has NOT eased our task. Rather, this example is being used to illustrate how a void function (a function that doesn't return anything) works. In particular, it's just as easy to directly call the print function in each of these instances instead of going through our defined printString function. One thing to note is that the printString function is quite flexible. We can give it any string we want, and any number of repetitions. On the first function call, our picture of memory in the printString function is as follows:

This will then print out two space characters. In the following function call, the picture of memory in the printString function is as follows:

In this manner, we can use the printString function in various ways to get our desired output result:

```
  *
 ***
*****
 ***
  *
```

Building on the printString function, we can create other functions that call it that print out different designs. Obviously the main above is fairly inefficient to print out the corresponding diamond it prints. Let's add some functions to make this task easier.

We can break down a diamond into two parts: a triangle with its point facing up and a triangle with its point facing down:

```
  *
 ***
*****

 ***
  *
```

Notice that the second triangle is "shifted" over one space from being left justified.

Thus, if we want a function to be able to carry out these tasks independently, the both functions need these pieces of information:

1) Number of Rows
2) The character we will use to print the triangle
3) Number of Spaces preceding each row from the left-justified position of the whole figure

Let's write a function that prints the top triangle:

```
def topTriangle(numRows, ch, leftSpaces):

    spaces = numRows-1

    for stars in range(1, 2*numRows, 2):
        printString(' ', leftSpaces)
        printString(' ', spaces)
        printString(ch, stars)
        printString('\n', 1)
        spaces -= 1
```

Here is how we designed the function:

The number of stars starts at 1 and ends at 2*numRows-1, so this is how we came up with the loop. It loops once for each row, and stars is equal to the number of stars for the row the corresponds to the loop iteration it's on. To handle spaces, we start this at numRows-1, which is the correct number of spaces for the first row. Then, for each loop iteration, we just subtract 1 from this variable.

Now, when printing, we have four sections to print: (1) the left spaces which, must appear for each line, (2) the regular spaces, (3) the regular repetitions of the character ch, (4) the newline character. We just stuck all of this in the loop, along with the necessary update to spaces.

Similarly, we can define a bottom triangle function:

```
def bottomTriangle(numRows, ch, leftSpaces):

    spaces = 0

    for stars in range(2*numRows-1, 0, -2):
        printString(' ', leftSpaces)
        printString(' ', spaces)
        printString(ch, stars)
        printString('\n', 1)
        spaces += 1
```

We use the same design, but spaces would start at 0 and the stars loop counts backwards from 2*numRows-1 and subtracts 2 each time, and at the end of each loop iteration, we add 1 to spaces, instead of subtracting.

Now that we have these two functions, we are ready to create a diamond function. We define carats to be the number of rows up to and counting the middle row.

```
def diamond(carats, ch):

    topTriangle(carats, ch, 0)
    bottomTriangle(carats-1, ch, 1)
```

Now, we can put together a whole program which prints out a diamond according to the user's specification, as follows:

```
# Arup Guha
# 2/29/2020
# Program to accompany function notes for COP 2930

def main():

    # Get what the user wants.
    rows = int(input("How many carats do you want your diamond to be?\n"))
    ch = input("What character do you want to print it with?\n")

    # Print it!
    print("Here you go:\n")
    diamond(rows, ch)
```

```
# Prints myStr n times.
def printString(myStr, n):
    print(myStr*n, end="")

# Prints a top triangle (point at top) of numRows using the character ch,
# padding with leftSpaces # of spaces on the left.
def topTriangle(numRows, ch, leftSpaces):

    # Spaces for first row.
    spaces = numRows-1

    # Loop through each row, loop index equals # of stars for each row.
    for stars in range(1, 2*numRows, 2):

        # We do left spaces follows by regular spaces.
        printString(' ', leftSpaces)
        printString(' ', spaces)

        # Now we print our stars and go to the next line.
        printString(ch, stars)
        printString('\n', 1)

        # For next row!
        spaces -= 1

# Prints a bottom triangle (point at bottm) of numRows using the character
# ch, padding with leftSpaces # of spaces on the left.
def bottomTriangle(numRows, ch, leftSpaces):

    # Spaces for the first row.
    spaces = 0

    # Stars on each row, counting down.
    for stars in range(2*numRows-1, 0, -2):

        # We do left spaces follows by regular spaces.
        printString(' ', leftSpaces)
        printString(' ', spaces)

        # Now we print our stars and go to the next line.
        printString(ch, stars)
        printString('\n', 1)

        # For next row!
        spaces += 1

# Prints a diamond of carats number of carats using the character ch.
def diamond(carats, ch):

    topTriangle(carats, ch, 0)
    bottomTriangle(carats-1, ch, 1)

# Run it!
main()
```