Introduction to Computers
Laboratory Manual

Experiment #6

Functions
What is Function?

A **function** is a block of organized, reusable code that is used to perform a single related action. **Functions** can be used to define reusable code and organize and simplify code.

As you already know, Python gives you many built-in functions like `print()`, `input()`, etc. But you can also create your own functions. These functions are called **user-defined functions**.

Defining a Function

A function definition consists of the function’s name, parameters, and body. The syntax for defining a function is as follows:

```python
def functionName(list of parameters):
    # Function body
```

Ex: The following function prints the statement "Hi, from IUG" 3 times.

```python
def print_hi():
    for i in range(3):
        print "Hi, from IUG"
```

Note that if we run the previous program it will print nothing. The previous code **defines** a function, but still we need to **invoke** that function to be executed.

Calling a Function

Calling (invoking) a function **executes the code in the function**.

In a function’s definition, you define what it is to do. To use a function, you have to **call** or **invoke** it. The program that calls the function is called a caller.

The following code invokes the method `print_hi()` which we defined previously:

```python
print_hi()
```

Note that any function must be defined before it can be called. Thus, the following code is correct:
Experiment #6: Functions

```python
def print_hi():
    for i in range(3):
        print "Hi, from IUG"

print_hi()
```

However, the following code is wrong and will not run correctly:

```python
def print_hi():
    for i in range(3):
        print "Hi, from IUG"

print_hi()
```

**Ex:** Write a function that takes a string as a parameter and prints that string 3 times to the screen. Then invoke the function to print "Hi, Amazing day!".

```python
def print_s(string):
    for i in range(3):
        print string

print_s("Hi, Amazing day!")
```

Note that if we called the later method without passing a parameter, we will get an error and the code will not run correctly.

Notes about functions:

- A function contains a header and body. The header begins with the `def` keyword, followed by the function’s name and parameters, and ends with a colon.
- A parameter is like a placeholder: When a function is invoked, you pass a value to the parameter. This value is referred to as an actual parameter or argument. Parameters are optional; that is, a function may not have any parameters.
- The function body contains a collection of statements that define what the function does.

💡 Functions with/without Return Values

There are two ways to call a function, depending on whether or not it returns a value.
A function that does not return a value is known as a **void function**. The call to void functions must be a statement. For example, the `print_hi` function does not return a value so the following call is a statement:

```python
print_hi()
```

A function that returns a value is known as a **value-returning function**, and if the function returns a value, a call to that function is usually treated as a value. For example, the `input()` function is a value-returning function, it returns the value entered by the user. The following code illustrates the idea.

```python
num = input("Enter a number: ")
```

Suppose the user entered the number 3, the previous code is now equivalent to this:

```python
num = 3
```

So, the interpreter substitutes the return value 3 for entire function invocation.

To write a value-returning function we use the `return` keyword to indicate the value to be returned to the caller.

**Ex:** Write a function that returns the maximum of two number.

```python
def max(num1, num2):
    if (num1 > num2):
        result = num1
    else:
        result = num2
    return result
```

Note that the `max` function is a value-returning function, so, the returned value should be stored in a variable to be used later. We can invoke the previous function as follows:

```python
larger = max(5, 8)
print(larger)
```

Another example of a call that is treated as a value is

```python
print(max(3, 4))
```
which prints the return value of the function call max (3, 4).

Ex: Write a function that prints the sum of two numbers.

```python
def sum(num1, num2):
    print(num1 + num2)

sum(3, 4)
```

Ex: Write a function that returns the sum of two numbers.

```python
def sum(num1, num2):
    return num1 + num2

print(sum(3, 4))
```

⚠️ Passing Arguments by Values

When you invoke a function with arguments, each argument's value is passed to the parameter in the function. **The value of an argument is passed to a parameter when invoking a function.** If the argument is a number or a string, the argument is not affected, regardless of the changes made to the parameter inside the function. The following code illustrates the idea:

```python
def increment(n):
    n += 1
    print("n inside the function is", n)

def main():
    x = 1
    print("Before the call, x is", x)
    increment(x)
    print("After the call, x is", x)

main()  # Call the main function
```

Before the call, x is 1

n inside the function is 2

After the call, x is 1
As shown in the output, the value of \( x \) (1) is passed to the parameter \( n \) to invoke the increment function. The parameter \( n \) is incremented by 1 in the function, but \( x \) is not changed no matter what the function does.

Positional and Keyword Arguments

A function’s arguments can be passed as positional arguments or keyword arguments. The power of a function is its ability to work with parameters. When calling a function, you need to pass arguments to parameters. There are two kinds of arguments: positional arguments and keyword arguments. Using positional arguments requires that the arguments be passed in the same order as their respective parameters in the function header. For example, the following function prints a message \( n \) times:

```python
def nPrintln(message, n):
    for i in range(n):
        print(message)
```

You can use \( \text{nPrintln(“hello”, 3)} \) to print \text{hello} three times. The \text{nPrintln(“hello”, 3)} statement passes \text{“hello”} to \text{message}, passes \text{3} to \text{n}, and prints \text{hello} three times. When we call a function like this, it is said to use positional arguments. The arguments must match the parameters in order, number, and compatible type, as defined in the function header.

You can also call a function using keyword arguments, passing each argument in the form name = value. For example, \( \text{nPrintln(n = 5, message = “good”)} \) passes \text{5} to \text{n} and \text{“good”} to \text{message}. The arguments can appear in any order using keyword arguments.

Default Arguments

Python allows you to define functions with default argument values. The default values are passed to the parameters when a function is invoked without the arguments.

The following example illustrates how to define functions with default arguments and how to call such functions:
def print_area(width=1.0, height=1.0):
    area = width * height
    print "width:", width, " height:", height, " area:", area

print_area()  # Default arguments width = 1 and height = 2
print_area(4, 2.5)  # Positional arguments width = 4, height = 2.5
print_area(height=5, width=3)  # Keyword arguments width
print_area(width=1.2)  # Default height = 2
print_area(height=6.2)  # Default width = 1

⚠️ Returning Multiple Values

The Python return statement can return multiple values. Python allows a function to return multiple values. The following program defines a function that takes two numbers and returns them in ascending order.

def sort(number1, number2):
    if number1 < number2:
        return number1, number2
    else:
        return number2, number1

n1, n2 = sort(10, 7)
print("n1 is", n1)
print("n2 is", n2)

And here is the output:

('n1 is', 7)
('n2 is', 10)
Lab Work

**Ex1:** Write a function that takes two numbers, x and y, and returns the sum of numbers from x to y.

Solution:

```python
def sum_range(x, y):
    sum = 0
    for i in range(x, y + 1):
        sum += i
    return sum
```

Now we can use `sum_range` function to sum numbers between 0-10, 1-100 and 100-150.

```python
print(sum_range(0, 10))
print(sum_range(1, 100))
print(sum_range(100, 150))
```

**Ex2:** Write a function that returns the absolute value of a number. If the function was called without any arguments, it should return the absolute value of 1.

Solution:

```python
def absolute_value(num=1):
    if num >= 0:
        return num
    else:
        return -num

print(absolute_value(2))
print(absolute_value(-4))
print(absolute_value())
```

**Ex3:** Write a function that prints the grade for a given score. Then use the function to print the score entered by user.

Solution:
def print_grade(score):
    if score >= 90.0:
        print('A')
    elif score >= 80.0:
        print('B')
    elif score >= 70.0:
        print('C')
    elif score >= 60.0:
        print('D')
    else:
        print('F')

score = input("Enter a score: ")
print_grade(score)

Ex4: Write a function isPrime that determines whether an integer is a prime number or not.

(A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself).

Solution:

def isPrime(num):
    if num < 2:
        return False
    for i in range(2, num):
        if (num % i) == 0:
            return False
    return True
Homework

1. Write \texttt{factorial(num)} function that computes the factorial of an integer.

2. Write a function \texttt{print\_prime\_numbers(num)} that prints all the prime numbers up to \texttt{num}.
   (hint: use \texttt{isPrime} function from the lab work.)

3. Using the following function header, write a function that computes the distance between two points.
   \begin{verbatim}
   def distance(x1, y1, x2, y2):
   \end{verbatim}

4. Complete the following two functions implementation:
   \begin{verbatim}
   # Converts from Celsius to Fahrenheit
   def celsiusToFahrenheit(celsius):
   
   # Converts from Fahrenheit to Celsius
   def fahrenheitToCelsius(fahrenheit):
   \end{verbatim}
   The formulas for the conversion are:
   \begin{align*}
   \text{celsius} &= (5/9) \times (\text{fahrenheit} - 32) \\
   \text{fahrenheit} &= (9/5) \times \text{celsius} + 32
   \end{align*}

5. Write a function that computes the sum of the digits in an integer. Use the following function header:
   \begin{verbatim}
   def sum_digits(n):
   \end{verbatim}
   For example, \texttt{sum\_digits(234)} returns 9, which is the sum of 2 + 3 + 4.

Good Luck

😊