Name suliman khan
ID 14282
Subject data sciences

Q1. a. Why Functions are used discuss in detail?

b. How arguments are used in function , write a simple program in Python

a. Why Functions are used discuss in detail

Ans :

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result. This way the functions used in the programming.

Functions are "self contained" modules of code that accomplish a specific task. Functions usually "take in" data, process it, and "return" a result. Once a function is written, it can be used over and over and over again. Functions can be "called" from the inside of other functions.

A function *is a rule which relates the values of one variable quantity to the values of another variable quantity, and does so in such a way that the value of the second variable quantity is uniquely determined by (i.e. is a* function *of) the value of the first variable quantity.*

Why do we Write Functions?

1. They allow us to conceive of our program as a bunch of sub-steps. (Each sub-step can be its own function. When any program seems too hard, just break the overall program into sub-steps!)
2. They allow us to reuse code instead of rewriting it.

3. Functions allow us to keep our variable namespace clean (local variables only "live" as long as the function does). In other words, function_1 can use a variable called i, and function_2 can also use a variable called i and there is no confusion. Each variable i only exists when the computer is executing the given function.
4. Functions allow us to test small parts of our program in isolation from the rest. This is especially true in interpreted langaues, such as Matlab, but can be useful in C, Java, ActionScript, etc.

Steps to Writing a Function

1. Understand the purpose of the function.
2. Define the data that comes into the function from the caller (in the form of parameters)!
3. Define what data variables are needed inside the function to accomplish its goal.
4. Decide on the set of steps that the program will use to accomplish this goal. (The Algorithm)

Function Workspace

Every function has its own **Workspace**. This means that every variable inside the function is only **usable** during the execution of the function (and then the variables go away).

Having a separate "workspace" for each function is critical to proper software engineering. If every function shared every variable in an entire program, it would be easy to inadvertently change the values of variables that you shouldn't. Further, it would be hard to remember what "names" have been used elsewhere, and coming up with new names to represent similar ideas would be challenging.

Additionally, the function can only "see" the information that is "passed" to it via parameters. Thus the only way information can get "in" to the function is by using parameters.

b. How arguments are used in function , write a simple program in Python

Defining a **Function** Here are **simple** rules to define a **function** in **Python**. **Function** blocks begin with the keyword def followed by the **function** name and parentheses (( )). Any input **parameters** or **arguments** should be placed within these parentheses. You can also define **parameters** inside these parentheses.

A simple program in python

```
def f(x, y):
    z = 2 * (x + y)
    return z


print("Program starts!")
a = 3
res1 = f(a, 2+a)
print("Result of function call:", res1)
a = 4
```

```
b = 7
res2 = f(a, b)
print("Result of function call:", res2)

Program starts!
Result of function call: 16
Result of function call: 22
```

2.

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

You can also send arguments with the *key* = *value* syntax.

This way the order of the arguments does not matter.

Example

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Q2. a. Why .upper(),.lower(),capitalize() and .swapcase() function are used

Ans :

The **upper**() **method** returns the uppercased string from the given string. It converts all lowecase characters to **uppercase**. If no lowercase characters exist, it returns the original string.

swapcase()

The string **swapcase**() **method** converts all uppercase characters to lowercase and vice versa of the given string, and returns it. Here string_name is the string whose cases are to be swapped. Parameter: The **swapcase**() **method** does not takes any parameter.

In this article, we will go for **capitalizing the characters** i.e. **conversion from lowercase to uppercase without using any function**. This article is based on the concept that how inbuilt function perform this task for us?

Example:

```
    Input:
    Hello world!

    Output:
    HELLO WORLD!
```

```
Python program to capitalize the character
# without using a function
st = input('Type a string: ')
out = ''
for n in st:
    if n not in 'abcdefghijklmnopqrstuvwqxyz':
        out = out + n
    else:
        k = ord(n)
        l = k - 32
        out = out + chr(l)
print('------->', out)
```

b. Write a program in which the discussed functions are used

Ans:

```
Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist = [1,2,3,4]; # This would assig new reference in
mylist
   print "Values inside the function: ", mylist
   return
```

```
# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

```
the out put

Values inside the function:  [1, 2, 3, 4]
Values outside the function:  [10, 20, 30]
```

Q3. a. What are the rules for defining the function?
Ans :

A function defined as a rule contains common calculations that can be broken out into logical pieces. Functions improve readability as the inputs and outputs are clearly defined. In addition, the majority of math variables used in a function do not need to be stored or used outside of the function. The math variables in a function are not stored in Math results, which improves system performance. Instead, the function is cached temporarily.

puts and outputs must be clearly **defined** and math variables mapped to the input and output of the **function** must be **defined** prior to the **function** call or a stack trace will occur.
Every **function** has a signature. This signature is the name, parameters and the data types being passed in and out

Functions Defined as Rules

- All math functionality available in transactions and business rule configuration is available within functions except LOG. Use LOG for the function's return value. Do not use LOG for math variables in the function itself.
- Functions may be global or they may be overridden. Overrides should be used as an exception to the more general higher level overrides.
    - The same functionality available in global functions is available in the override. Inputs and outputs must be clearly defined and math variables mapped to the input and output of the function must be defined prior to the function call or a stack trace will occur.
    - Every function has a signature. This signature is the name, parameters and the data types being passed in and out. There will be one signature that transcends all the overrides. All overrides are required to match a global signature.
- Functions are often chosen over CopyBooks for specific calculations since they offer increased readability.

- Functions can be defined at different levels. Container components using them will establish a context that finds the best matching level to execute for each individual function.

    - If a function is used multiple times in a transaction/business rule it will be resolved and compiled once, no matter how many times it is used in that transaction/rule.

Steps to Call a Function Defined as a Rule

1. Right-click on the XML file to call the function in.
2. Select **Check out**.
3. Open the Math pane.
4. From the Palette window, scroll down to the folder labeled **rule** and drag and drop **FunCall** onto the Configuration Area.

5. Name the math variable.
6. Click the **Data Type** field to select a data type from the drop down box.
7. Select whether to **LOG** or **Round** the function's output result.
8. Click the **Functions** field to select the function to call from the **Functions** drop-down box.
9. In the Input section, click the **Value** drop-down box to select each parameter's value.
10. In the Output section, click the **Value** drop down box to select each parameter's value.
11. Check-in the XML file to save the changes.

b. Write a suitable program of our defined function in Python

Ans:

In Python, a function is a group of related statements that performs a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes the code reusable.

Example of a function

```
def greet(name):
```

```
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
```

Q4. a. What are the rules for defining the function and Parameter passing to the function?

Ans :

Arguments are passed by **value**; that **is**, when a **function is** called, the **parameter** receives a copy of the **argument's value**, not its address. ... However, because **arguments can** be addresses or pointers, a **function can** use addresses to modify the values of variables defined in the calling **function**.

C functions exchange information by means of parameters and arguments. The term *parameter* refers to any declaration within the parentheses following the function name in a function declaration or definition; the term *argument* refers to any expression within the parentheses of a function call.

The following rules apply to parameters and arguments of C functions:

- Except for functions with variable-length argument lists, the number of arguments in a function call must be the same as the number of parameters in the function definition. This number can be zero.
- The maximum number of arguments (and corresponding parameters) is 253 for a single function.
- Arguments are separated by commas. However, the comma is not an operator in this context, and the arguments can be evaluated by the compiler in any order. There is, however, a sequence point before the actual call.
- Arguments are passed by value; that is, when a function is called, the parameter receives a copy of the argument's value, not its address. This rule applies to all scalar values, structures, and unions passed as arguments.
- Modifying a parameter does not modify the corresponding argument passed by the function call. However, because arguments can be addresses or pointers, a function can use addresses to modify the values of variables defined in the calling function.
- In the old style, parameters that are not explicitly declared are assigned a default type of `int` .
- The scope of function parameters is the function itself. Therefore, parameters of the same name in different functions are unrelated.

b. Write a suitable program of our defined function by parameter passing in Python?

Ans :

```
Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print str
   return;

# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

When the above code is executed, it produces the following result −

```
I'm first call to user defined function!
Again second call to the same function
```

Q5. a. What are return values to a Function discuss in detail?

Ans :

The `return` statement is used to exit a function and go back to the place from where it was called.

Syntax of return

```
return [expression_list]
```

This statement can contain an expression that gets evaluated and the value is returned. If there is no expression in the statement or the `return` statement itself is not present inside a function, then the function will return the `None` object.

For example:

```
>>> print(greet("May"))
Hello, May. Good morning!
None
```

When you create a new function that returns a string value, the Script Manager places the following function beginning line into your script file:

String Function MyFunction ()

The "string" key word that precedes the "function" key word tells you that the MyFunction function returns a string value to the calling script or user-defined function.

The Return Statement

You use the return statement to send a value back to the calling script or user-defined function. This key word tells JAWS to return the specified value to the calling script or function. You can return the value as a literal value or within a variable. The syntax of a return statement that sends a string value stored in a variable called sText back to the calling script or user-defined function follows:

Return sText

A return statement that returns a string of text, "This is a string", as a literal follows:

Return "this is a string"

When a function returns a value to the calling script or user-defined function, you must store that value in either a local or global variable. You can then use that variable to make decisions on what the calling script or user-defined function should do next. An example of storing a string value returned by a function called MyFunction in a local variable follows:

b. Write a suitable program of a Function with returning value?

Ans :

```
def absolute_value(num):
    """This function returns the absolute
    value of the entered number"""

    if num >= 0:
        return num
    else:
        return -num


print(absolute_value(2))

print(absolute_value(-4)
```

Output

```
2
4
```