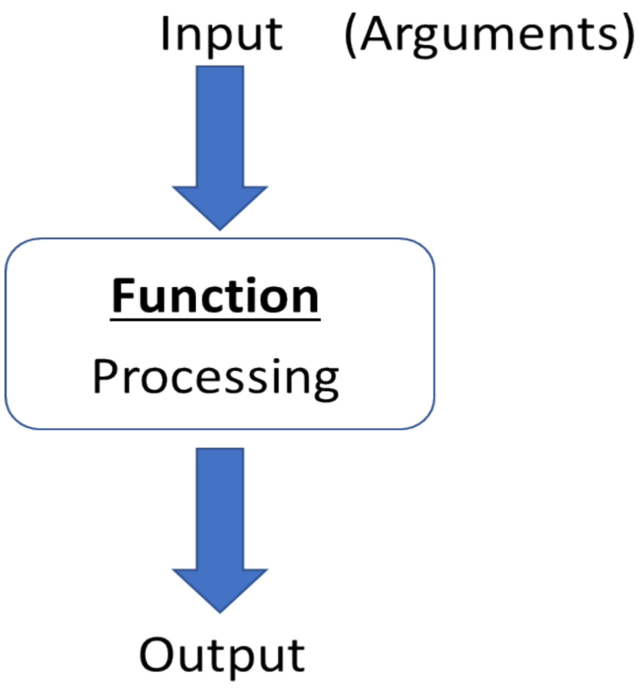
PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions: Definition and Call

Prof. Sindhu R Pai
PCPS Theory Anchor - 2024
Department of Computer Science and Engineering

What are functions?

- A function is a **self contained block of code** that performs a specific task
- Functions ideally **take input**, performs a set of **operations** and **returns an output** to the program that invoked it.



Advantages of functions

- Reducing **duplication** of code and the **complexity** of the program through **modularity.**
- Improves the **readability**, enhances the **clarity of the program.**
- Promotes **reuse of code.**
- **Debugging and maintenance** becomes easier

Functions - Types

Types of functions in Python programming:

- **Built-in functions or Pre-defined** – Standard library functions.

- **User defined functions** - Defined by user for specific application in order to reduce complexity of large programs.

Built-in Functions

Built-in functions or Pre-defined – These functions are built into the Python interpreter and can be used/called without the need for additional code.

Examples:

- print() – Outputs a message to the console or standard output device.
- Input() - Takes user input from the console or standard input device.
- len() - Returns the length of an object, such as a string, list, or tuple.
- type() - Returns the data type of an object.
- sorted() - Sorts a list or sequence in ascending order.
- abs() - Returns the absolute value of a number.

User Defined Functions

- **User defined functions** – These are the functions defined by the user to aid them in **achieving a specific goal**. The **Python interpreter** also supports the user-defined functions created as per the user requirement.

The main purpose of these functions is to help in **organizing the programs into logical segments** that work together to solve a specific part of our problem.

Definition:

- A function is a **block of code** which only runs when it is called.
- Can pass data, known as **parameters (optional),** into a function.
- A function can return data as a result.

User Defined Functions

A function has two parts – **leader and suite**

- The leader starts with the **keyword *def*** then the function name
- **function name is an identifier** that starts with alphabets [a- z or A-Z] or _ and then followed by any number of letter of English or _ or digit.
- followed by a pair of round parentheses then a colon.
- **suite** follows – suite can have any valid statement of Python including another function definition.
- All the statements within the function must be **indented.**
- Finally, use the **return (optional) keyword** to return the output of the function.
- Function must be defined preceding their usage in the program code.

Syntax:

```
def  function_name(parameters):
          #suite
```

Processing of Functions

- When the function is defined, header or the leader is processed; the user is provided a name or a handle which is the same as the function name.

- A function entity with the function name in the definition along with the suite is stored.

- Each entity in Python has a reference count.

- At this point of translation, only leader is processed and the suite is not processed.

Function: Call

- **Name of the function followed by parentheses** causes a function call

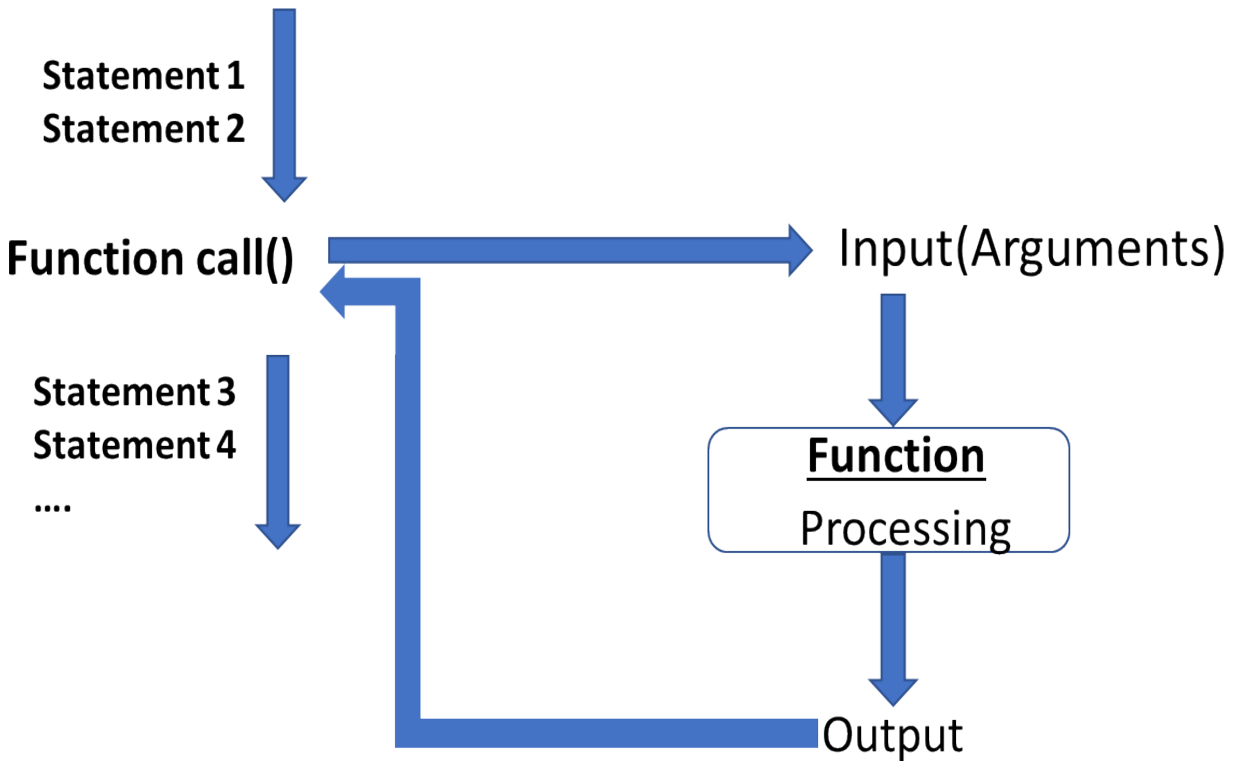
- This results in transfer of control to the **leader of the function** and then the suite is executed

- After that, the control comes back to the point after the function call in the user’s code.

- A **pair of round parentheses ()** is called a function call operator.

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Function: Call



Function Call: Activation record

When the function call is made, an **activation record** is created which will have

- **Parameters:-** nothing but the arguments.
- **Local variables:-** variables created with in the suite of the function.
- **Return address:-** location to which the control of the program should be transferred once the function terminates.
- **Temporary variables:-** unnamed variables required by the translator
- **Return value:-** value to be passed back to the caller.

The activation record is created for every call of the function.

At the end of the function, if no other callable can refer to the activation record, it will be removed.

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

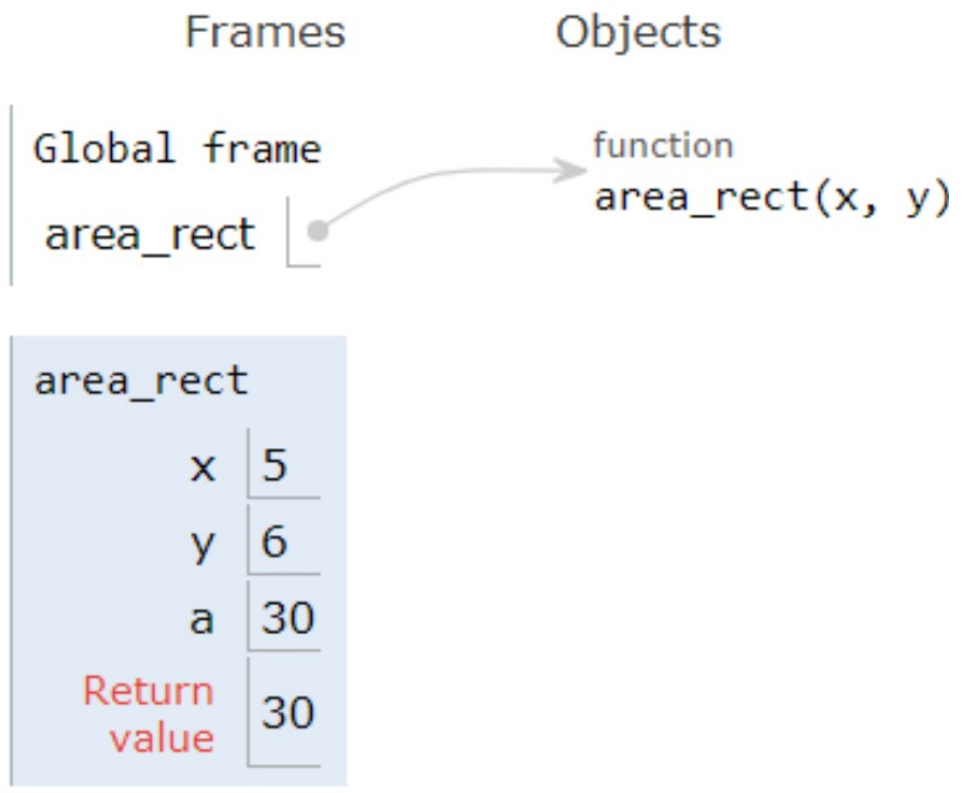
Function Call:

Example 1:-

```
def area_rect(x,y) :
        a=x*y
        return a
a=area_rect(5,6)
print("area =",a)
```

Output:

```
area = 30
```



Function Overloading:

Function overloading is the ability to have multiple functions with the same name but with different signatures/implementations.

Python **does not support function overloading**. When we define multiple functions with the same name, the new function always overrides the prior function.

Example:-

```
def add(a,b):
      p=a+b
      print(p)
def add(a,b,c):
     p=a+b+c
     print(p)
# add(2,3) this will not run, it gives error
add(1,2,3)
```

User - Defined Functions : Example

Example 1:

```
#program to display a greeting message
def display():
    print("hello")
    print("python")
    print("program")
display()
```

Output:

```
hello
python
program
```

Function: Call

Example 2:

```
def display() :
        print("hello") ;print("python")
display()

display1=display            # assigning the function entity another name
# at this point the reference count of the function entity is up by 2
del display                 # ref count reduces by 1
display1()                  # still works
```

Output:

```
hello
python
```

Multiple Return statements

- **Functions can have multiple return statements**, but any statement after the 1st return statement becomes part of the unreachable code.
- The python interpreter is forgiving in terms of letting its programmers **make these kind of errors.**
- **Example:-**
- def example():

```
def example():
  print(“an example function”)
  return #function ends here
  print(“after return”) #unreachable
  return ‘Hello’ #unreachable
```

Function Call: Returning multiple values

When a collection of values is returned from called function, the interpreter puts it together into a tuple and returns it to the calling program.

Example 3-:

```
def add():
        a = 12
        b = 13
        s = a+b
        return s,a,b    # becomes an unnamed tuple
sum = add()
print(type(sum))    # <class 'tuple'>
print(sum)          # (25, 12, 13)
```

Output:

```
<class 'tuple'>
(25, 12, 13)
```

Function Call: Value returning and Non-Value returning Functions

Non-Value Returning Functions – Functions are not required to return a value back to the caller. **By default, it returns None.**

Example 2:-

```
def multiply_numbers(a,b):
   product=a*b
   print(product)
```

Value Returning Functions – Functions that are required to return a value back to the caller.

Example 2:-

```
def multiply_numbers(a,b):
   product=a*b
   return product
```

```
n1=10
n2=20
print("Product=",multiply_numbers(n1,n2))
```

Function Call: Categories of Functions

1. No arguments: No return value

```
def add():
        a = 10
        b = 20
        print(a+b)
add()
```

Output: 30

2. No arguments: with return value

```
def add()
   a=10
   b=20
   return a+b
sum = add()
print(sum)
```

Output: 30

Function Call: Categories of Functions

3. With arguments: No return value

```
def add(a,b):
        print(a+b)
add(10,20)
```

Output:

```
30
```

4. With arguments: With return value

```
def add(a,b):
    return a+b
sum = add(10,20)
print(sum)
```

Output:

```
30
```

THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CCBD & CDSAML, PESU
Prof. Sindhu R Pai – sindhurpai@pes.edu
Dr. Chetana Srinivas

Ack: Teaching Assistant – Advaith Sanil Kumar