Function

Function is a **reusable** block of code that performs a **specific** task.

Task

```
Statement 1
Statement 2
    ...
Statement n-1
Statement n
```

Addition

```
a = 5     # Create integer object and assign it to a
b = 10    # Create integer object and assign it to b
C = a+b   # Add two object and assign result to c
```

What if you need to perform the addition at **multiple parts/locations** in your program?

How to write function in python?

```
def <name>(arg1, arg2, …, argN):
    <statement 1>
    <statement 2>
       ....
    <statement M>
```

User-defined Function

Built-in Function:
E.g, print(), input(), len()

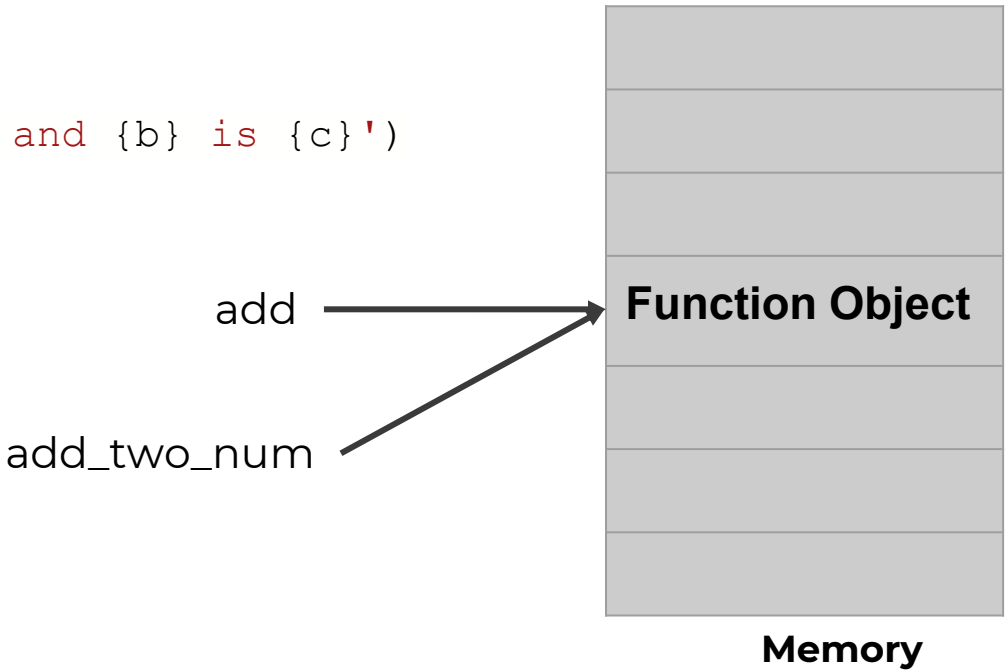
Call function

```
<name>(value1, value2, ..., valueN)
```

How to write function in python?

```
def <name>(arg1, arg2, …, argN):
    <statement 1>
        ....
    <statement M>
```

```
def add(a, b):
    c = a+b
    print(f'The addition of {a} and {b} is {c}')
```



Code Demo in Jupyter Notebook

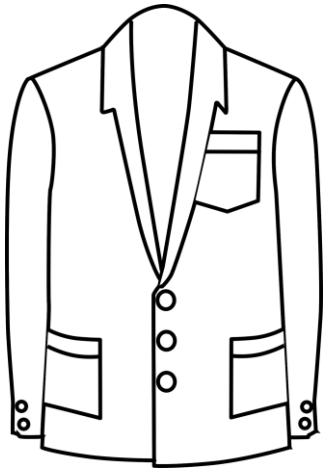
But what if we want to return the value?

```
def add(a, b):
    c = a+b
    return c


result = add(2,3)
print(result)
```

Code Demo in Jupyter Notebook

Advantages/Features of function

1. Reusability



Advantages/Features of function

2. Encapsulation/Abstraction

Square Root

$$\sqrt{5}, \sqrt{6}, \sqrt{7} \quad ?$$

What should we know about function?

- **its name**
- **what it does?**
- **its argument**
- **its return**

Advantages/Features of function

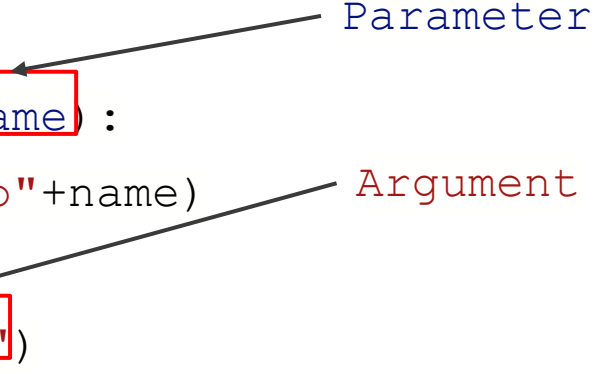
3. Procedural Decomposition

```
take_order()
prepare_order()
make_pizza()
make_burger()
    ...
serve_item()
```

Arguments

Arguments allows us to pass information into function

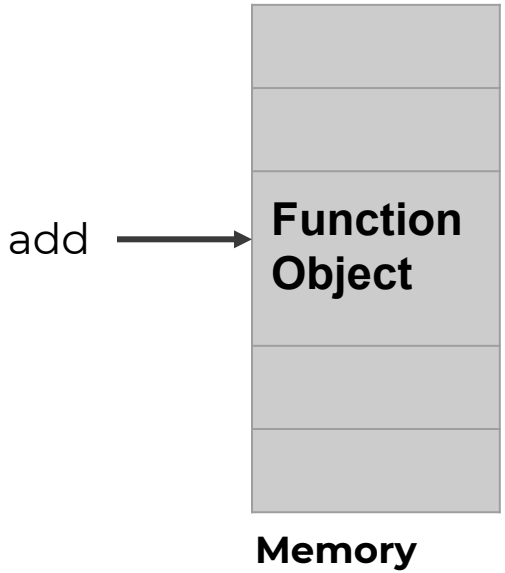
```
# Greeting Function

def greeting(name):
    print("Hello"+name)

greeting("John")
```



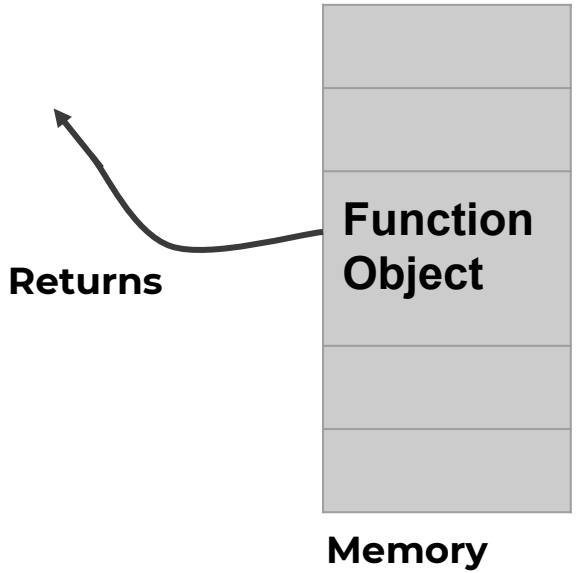
Lambda Function

Lambda function creates a function object and returns it.

```
def add(a, b):
    return a+b
```



Lambda Function



How to define a lambda function?

Syntax of Lambda Function:

```
lambda arguments: expression
lambda arg1, arg2, arg3, ..., argN : expression
```

```
double = lambda x: x * 2
print(double(3))
```

```
def double(x):
    return x * 2
```

Code Demo in Jupyter Notebook

Recursion

Recursive function are those function that call itself to solve a problem.

```
def some_function(a):
    <statements>
    some_function(x)
    <statements>
```

```
some_function(10)
    some_function(10)
        some_function(10)
            some_function(10)
                some_function(10)
                    some_function(10)
                        ...
```

Factorial using recursive function

$$n! = n * (n-1) * (n-2) \ldots .3 * 2 * 1$$

$$5! = 5 * 4 * 3 * 2 * 1$$

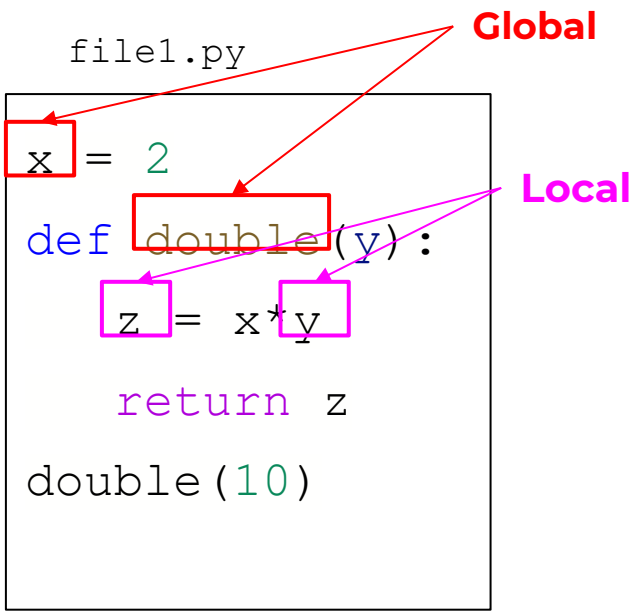
```
def factorial(n):
    if n == 1:
        return n
    else:
        return n*factorial(n-1)
```

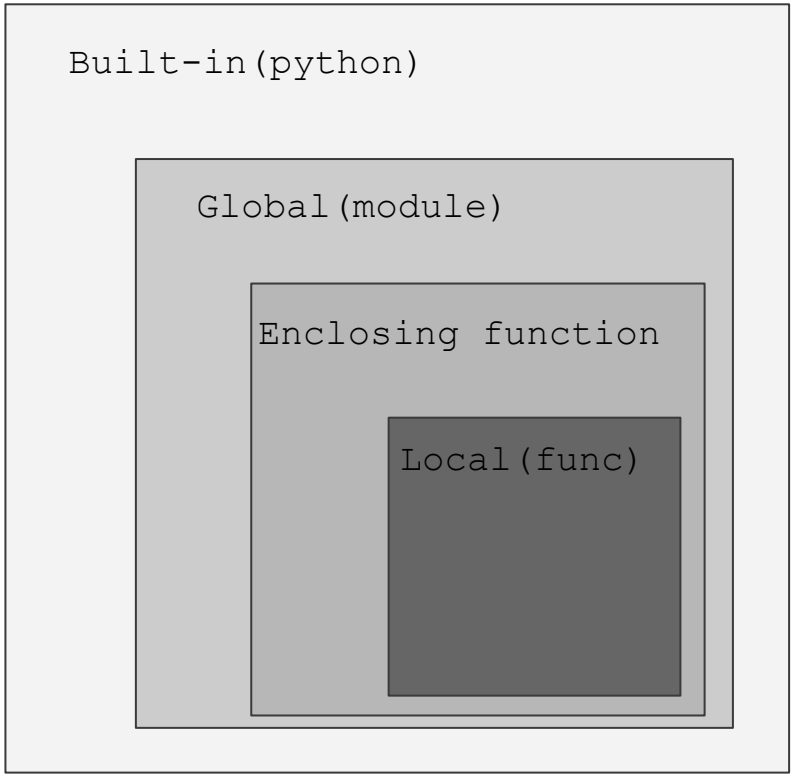
```
factorial(5)
5*factorial(4)
5*4*factorial(3)
5*4*3*factorial(2)
5*4*3*2*factorial(1)
5*4*3*2*1
```

Code Demo in Jupyter Notebook

Variable Scope

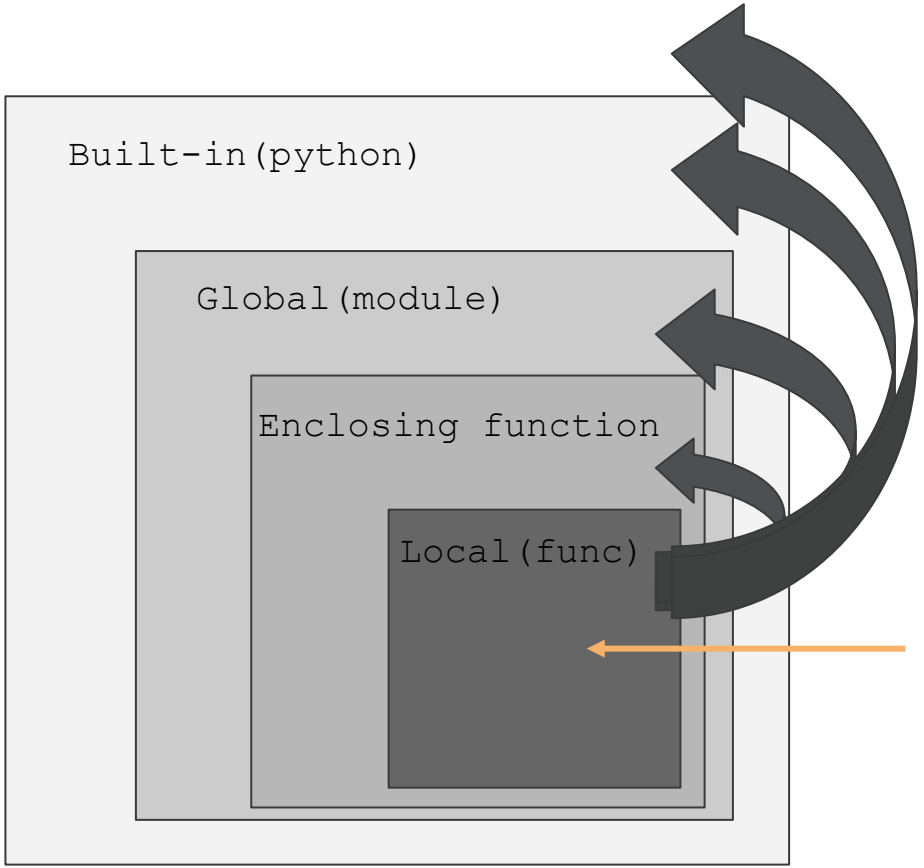
Scope of a variable is the region inside which the variable is created.





Code Demo in Jupyter Notebook

Name Resolution: LEGB and `global`



Error:
Variable not defined

Code Demo in Jupyter Notebook

Key Takeaways

1. Function creation syntax:

```
def <name>(arg1, arg2, …, argN):
    <statements>
```

1. Argument types: **Arbitrary, Keyword, Arbitrary Keyword, Default.**
2. **Lambda function** syntax: `lambda arguments: expression`
3. A **recursive** function **call itself**.
4. Scope of a variable: **Local, enclosing function, global, built-in**.