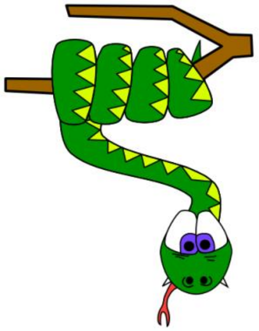


Functions

Floating Point Representation

- The **float** type in Python represents the floating point number. Float is used to represent real numbers and is written with a decimal point dividing the integer and fractional parts.
- For example, 97.98, 32.3+e18, -32.54e100 all are floating point numbers.
- Python float values are represented as 64-bit double-precision values.

Floating Point Representation

- The maximum value any floating-point number can be is approx 1.8×10^{308}. Any number greater than this will be indicated by the string inf in Python.

Example :

```
# Python code to demonstrate float values.
print(1.7e308)
# greater than or equal to 1.8 * 10^308
# will print 'inf'
print(1.8e308)
```

Floating Point Representation

- Floating-point numbers are represented in computer hardware as base 2 (binary) fractions.
- For example, the decimal fraction 0.125 has value 1/10 + 2/100 + 5/1000,
- and in the same way the binary fraction 0.001 has value 0/2 + 0/4 + 1/8. These two fractions have identical values, the only real difference being that the first is written in base 10 fractional notation, and the second in base 2.

Floating Point Representation

- **float.as_integer_ratio() :**
- Returns a pair of integers whose ratio is exactly equal to the actual float having a positive denominator.

```
d = 3.5
b = d.as_integer_ratio()
print(b)
print(b[0], "/", b[1])
```

Output:

```
(7, 2)
7 / 2
```

Floating Point Representation

- **float.is_integer() :**

Returns True in case the float instance is finite with integral value, else, False.

```
# using is_integer
print((9.0).is_integer())
print((-5.0).is_integer())
print((4.8).is_integer())
print((6.67).is_integer())
```

Output:

```
True
True
False
False
```

Function

- A function can be defined as the organized block of reusable code which can be called whenever required.
- The function contains the set of programming statements(lines) .
- A single python file can contain multiple function.
- Function can be 2 types:
- 1.Predefine function
- Ex: print(),range() etc
- 2.UserDefine function
- Ex: price_calculator() ,display() etc

Need of Function

- Code reuse is one of the most prominent reason to use function .Large programs usually follow the DRY(Don’t Repeat Yourself)principle means don’t repeat yourself principle.

- The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.

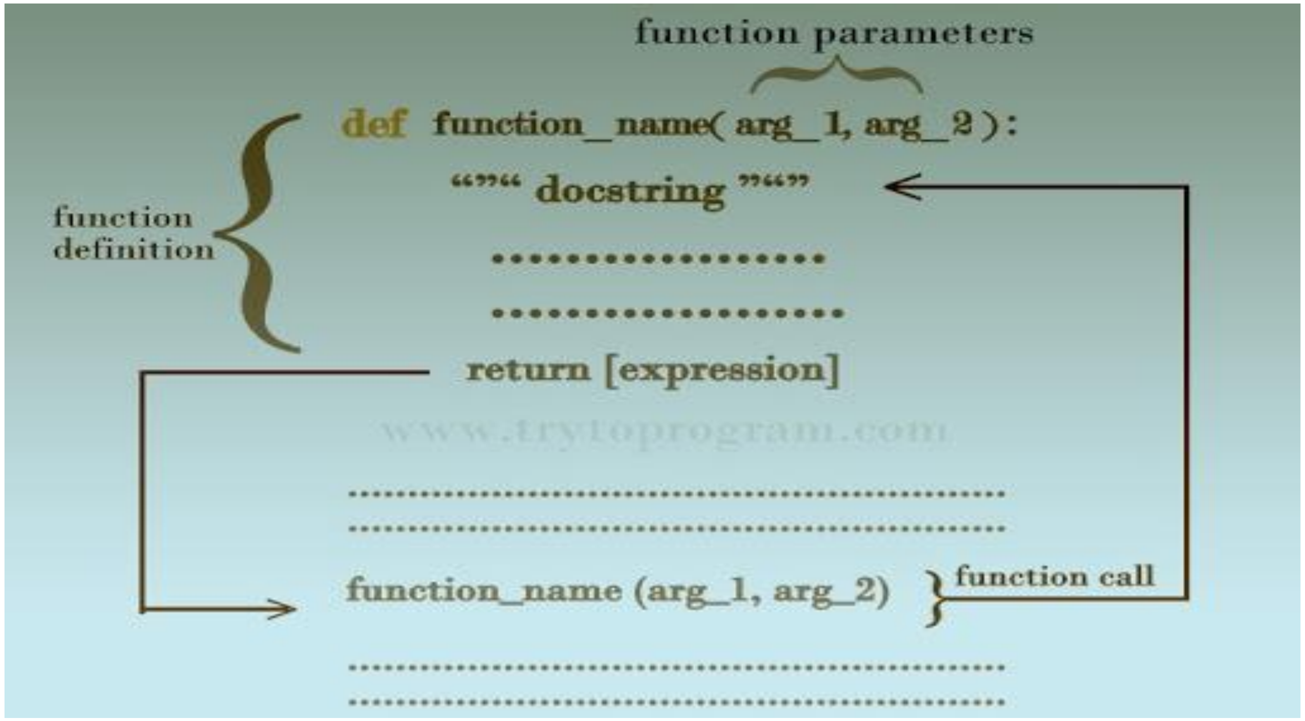
Function…

- **Creating and defining function:**

```
def   my_function(parameters):
       “ documentation”
      statements
      return <expression>
```

Function…

- **Creating and defining function:**



Function…

- **function without parameters**

Ex:1

```
def   add( ):
      a=5
      b=7
      c=a+b
      print(c)
add()
```

Ex:2

```
def   average_product():
       price=230
       product="fruits"
      result=price*product
      print(result)
average_product()
```

Function...

- **Function with parameter:**

```
#function Deffination
def calculate_sum(data1,data2):
    result_sum=data1+data2
    return result_sum

#Function calling


result=calculate_sum(24,45)
print(result)


#Other  statements
```

Scope Rule of variable’s

- In python, the variables are defined with the two types of scopes.

1. Global variables
2. Local variables

- The variable defined outside any function is known to have a global scope whereas the variable defined inside a function is known to have a local scope.

Example:

```
var1="I am global variable"
def my_function():
    var2="I am local variable"
    print(var2)
my_function()
print(var1)
```

Using the global Statement

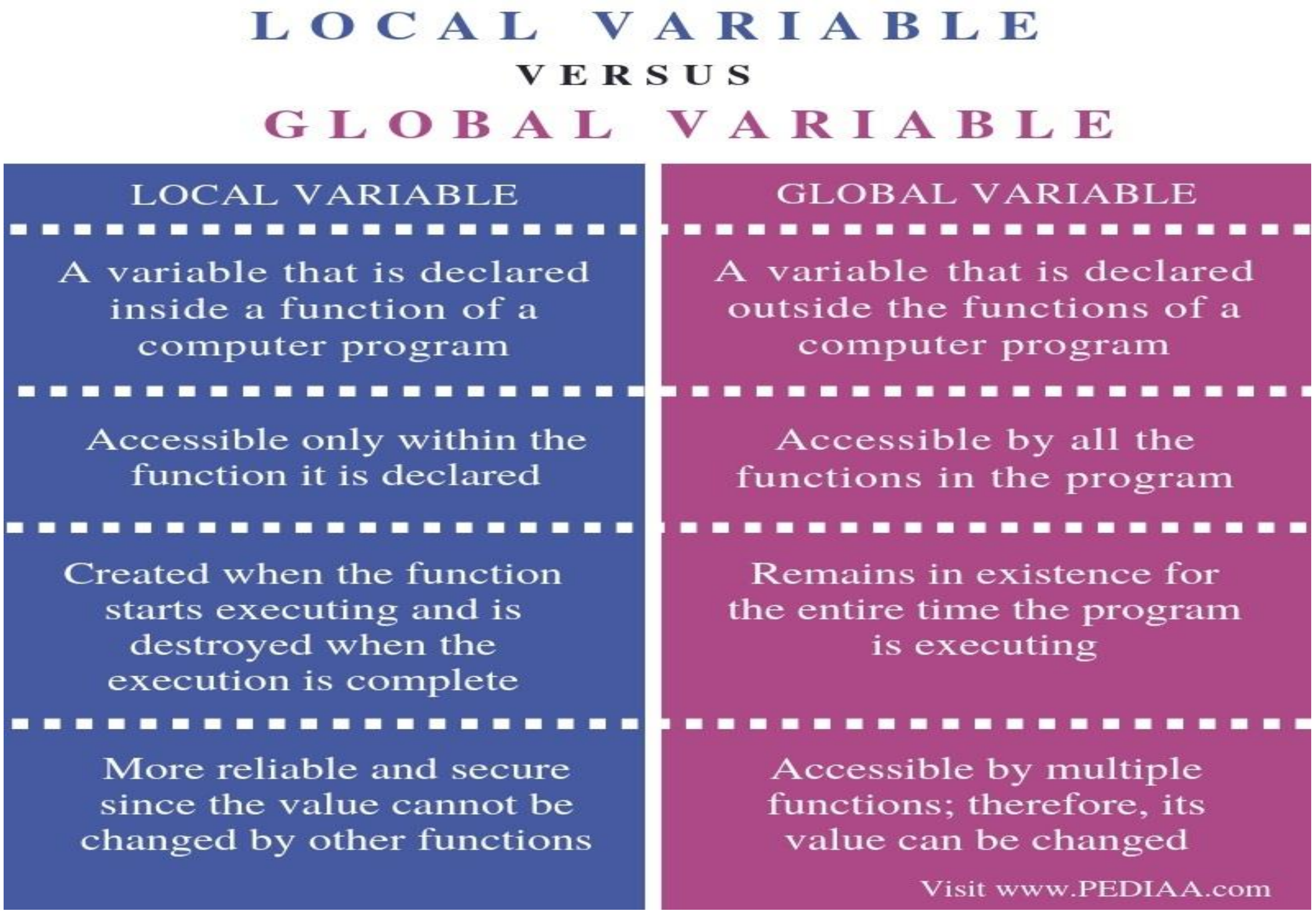
- **global Statement:**

- To define a variable inside a function as global ,you must use the global statement . Where global is keyword.

Example:

```
var1="I am global variable"
def my_function():
    global var2
    var2="I am also global  variable"
my_function()
print(var1)
print(var2)
```

Function…



LOCAL VARIABLE

VERSUS

GLOBAL VARIABLE

LOCAL VARIABLE	GLOBAL VARIABLE
A variable that is declared inside a function of a computer program	A variable that is declared outside the functions of a computer program
Accessible only within the function it is declared	Accessible by all the functions in the program
Created when the function starts executing and is destroyed when the execution is complete	Remains in existence for the entire time the program is executing
More reliable and secure since the value cannot be changed by other functions	Accessible by multiple functions; therefore, its value can be changed

Visit www.PEDIAA.com

Function Argument

- You can call a function by using the following types of formal arguments –

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

Required Argument

- Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

Example:

```
def  my_function(a,b):
    c=a+b
    print(c)
my_function(3,4)
```

keyword **Argument**

- Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

Example:

```
def display(str1,int_x,float_y):
    print("the string is ",str1)
    print("the integer is ",int_x)
    print("the float is ",float_y)
display(float_y=3.4,str1="CE_II_A",int_x=3)
```

Default **Argument**

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. The following example gives an idea on default arguments, it prints default age if it is not passed −

```
# Function definition is here
def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   print("Name: ", name)
   print("Age ", age)

# Now you can call printinfo function
printinfo("miki")
printinfo("geny",45 )
```

Variable Length **Argument**

- In some situation ,it is not known in advance how many arguments will be passed to a function .In such cases,python allows programmers to make function calls with any number of arguments.
- When we use variable length argument then function definition uses An astrisk (*) before the parameter name.

Example:

```
def func(name,*fav_subject):
    print(name)
    for subject in fav_subject:
        print(subject)
func("goransh","maths","python programming","matlab")
func("Richa","Art","Java programming","C‟,”maths”)
func("Krish","English","Android")
```

Difference between *args and **kwargs

- The special syntax **args* in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-keyworded, variable-length argument list.

- **kwargs works just like *args, but instead of accepting positional arguments it accepts keyword (or **named**) arguments.

- For more detail use following link :
- https://www.geeksforgeeks.org/args-kwargs-python/

Difference between *args and **kwargs

```
def func(name,**fav_subject):
    print(name)
    for subject in fav_subject.values():
        print(subject)
func("goransh",one="maths",two="python
  programming",three="matlab")
func("Richa",one="Art",two="Java programming",three="C")
func("Krish",one="English",two="Android",three="DS")
```

Python lambda (Anonymous Functions)

- In Python, anonymous function means that a function is without a name.

- the *lambda* keyword is used to create anonymous functions. It has the following syntax:

result=lambda arguments: expression

Where lambda is keyword, argument is number of parameter and expression is programming expression that need to evaluate according to parameter.

- This function can have any number of arguments but only one expression, which is evaluated and returned.

Python lambda (Anonymous Functions)

- Example 1:

```
g =lambda x: x*x*x
print(g(7))
```

Output:343

Example 2:

```
x = lambda a, b : a * b
print(x(5, 6))
```

Output:30