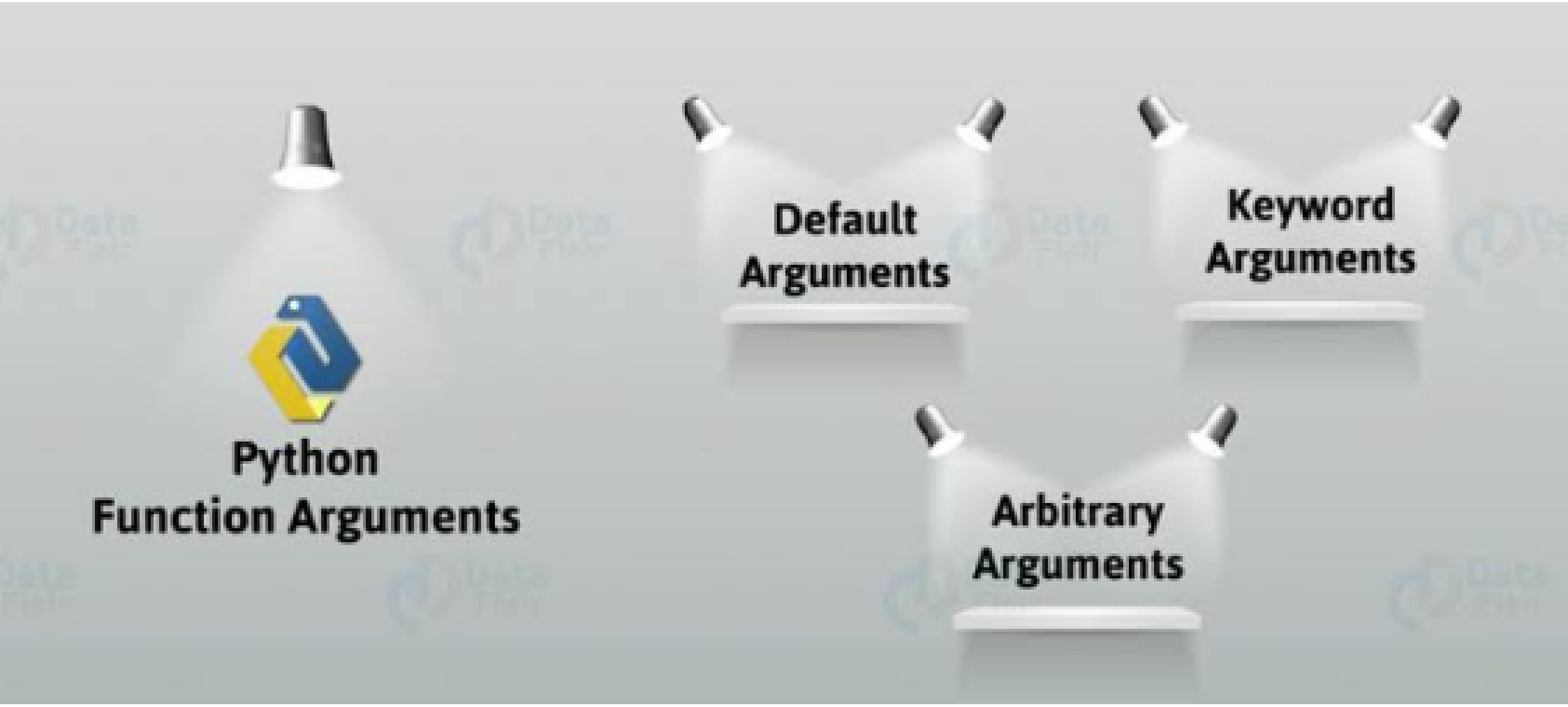
Python


Default
Arguments
Keyword
Arguments
Python
Function Arguments
Arbitrary
Arguments

Types of Python Function Arguments

Python supports various types of arguments that can be passed at the time of the function call. In Python, we have the following 4 types of function arguments.

- **Default argument**
- **Keyword arguments (named arguments)**
- **Positional arguments**
- **Arbitrary arguments** (variable-length arguments *args and **kwargs)

Default Arguments

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments.

```
# Python program to demonstrate
# default arguments
def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)


# Driver code (We call myFun() with on
# argument)
myFun(10)
```

Output

```
x:  10
y:  50
```

Keyword Arguments

The idea is to allow the caller to specify the argument name with values so that the caller does not need to remember the order of parameters.

```
# Python program to demonstrate Keyword Arguments
def student(firstname, lastname):
    print(firstname, lastname)


# Keyword arguments
student(firstname='Geeks', lastname='Practice')
student(lastname='Practice', firstname='Geeks')
```

Output

```
Geeks Practice
Geeks Practice
```

Positional Arguments

We used the Position argument during the function call so that the first argument (or value) is assigned to name and the second argument (or value) is assigned to age. By changing the position, or if you forget the order of the positions, the values can be used in the wrong places, as shown in the Case-2 example below, where 27 is assigned to the name and Suraj is assigned to the age.

```
def nameAge(name, age):
    print("Hi, I am", name)
    print("My age is ", age)


# You will get correct output because
# argument is given in order
print("Case-1:")
nameAge("Suraj", 27)
# You will get incorrect output because
# argument is not in order
print("\nCase-2:")
nameAge(27, "Suraj")
```

Output:

```
Case-1:
Hi, I am Suraj
My age is  27

Case-2:
Hi, I am 27
My age is  Suraj
```

Arbitrary Keyword Arguments

In Python Arbitrary Keyword Arguments, *args, and **kwargs can pass a variable number of arguments to a function using special symbols. There are two special symbols:

- *args in Python (Non-Keyword Arguments)
- **kwargs in Python (Keyword Arguments)

How to Use *args in Python

`*args` allows us to pass a variable number of non-keyword arguments to a Python function. In the function, we should use an asterisk (`*`) before the parameter name to pass a variable number of arguments.

```
def add(*args):
    print(args, type(args))

add(2, 3)
```

Output:

```
(2, 3) <class 'tuple'>
```

Thus, we're sure that these passed arguments make a tuple inside the function with the same name as the parameter excluding `*`.

Now let's rewrite our `add()` function with a variable number of arguments.

```
def add(*numbers):
    total = 0
    for num in numbers:
        total += num
    return total


print(add(2, 3))
print(add(2, 3, 5))
print(add(2, 3, 5, 7))
print(add(2, 3, 5, 7, 9))
```

Output:

```
5
10
17
26
```

Note that the name of the argument need not necessarily be `args` – it can be anything. In this case it's `numbers`. But it's generally a standard way to use `*args` as the name.

How to Use **kwargs in Python

`**kwargs` allows us to pass a variable number of keyword arguments to a Python function. In the function, we use the double-asterisk (`**`) before the parameter name to denote this type of argument.

```
def total_fruits(**kwargs):
    print(kwargs, type(kwargs))


total_fruits(banana=5, mango=7,
apple=8)
```

Output:

```
{'banana': 5, 'mango': 7, 'apple': 8}
<class 'dict'>
```

Thus we see that the arguments, in this case, are passed as a dictionary and these arguments make a dictionary inside the function with name same as the parameter excluding `**`.

Now, let's complete the `total_fruits()` function to return the total number of fruit.

```
def total_fruits(**fruits):
    total = 0
    for amount in fruits.values():
        total += amount
    return total


print(total_fruits(banana=5, mango=7,
apple=8))
print(total_fruits(banana=5, mango=7,
apple=8, oranges=10))
print(total_fruits(banana=5, mango=7))
```

Output:

```
20
30
12
```

Note that the name of the argument need not necessarily be `kwargs` – again, it can be anything. In this case, it's `fruits`. But it's generally a standard way to use `**kwargs` as the name.