PYTHON
LECTURE 21

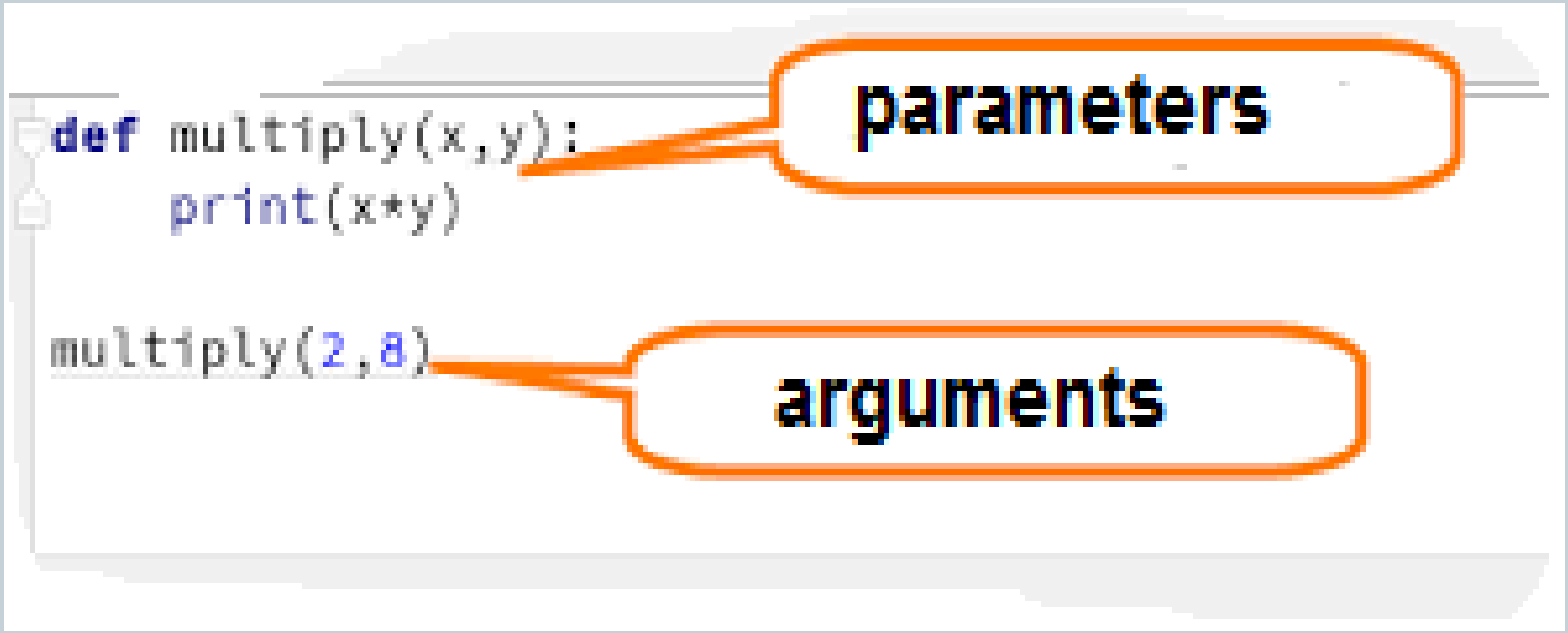
Today's Agenda

- **User Defined Functions-II**
  - Arguments V/s Parameters
  - Types Of Arguments

Parameters V/s Arguments?

- A lot of people—mix up **parameters** and **arguments**, although they are slightly different.
- A **parameter** is a variable in a **method definition**.
- When a method is called, the **arguments** are the data we pass into the method's **parameters**.

Parameters V/s Arguments?



```
def multiply(x,y):
    print(x*y)

multiply(2,8)
```

Types Of Arguments

- In **Python** , a function can have **4** types of arguments:
  - **Positional Argument**
  - **Keyword Argument**
  - **Default Argument**
  - **Variable Length Argument**

Positional Arguments

- These are the arguments passed to a function in correct **positional order**.

- Here the number of **arguments** in the call must exactly match with number of **parameters** in the function definition

Positional Arguments

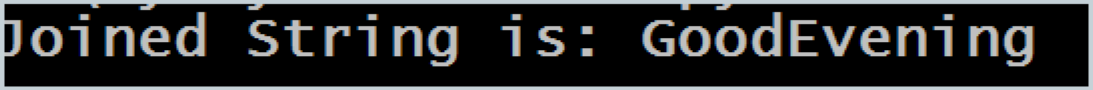
```
def attach(s1,s2):
    s3=s1+s2
    print("Joined String is:",s3)

attach("Good","Evening")
```

These are called

POSITIONAL ARGUMENTS

Output:



```
Joined String is: GoodEvening
```

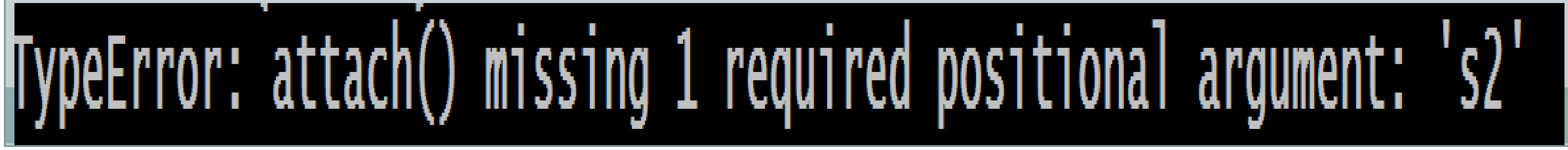
Positional Arguments

- If the **number of arguments** in call do not match with the **number of parameters** in function then we get **TypeError**:

```
def attach(s1,s2):
    s3=s1+s2
    print("Joined String is:",s3)

attach("Good")
```

Output:



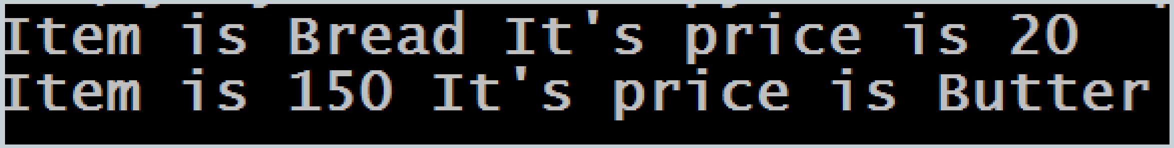
```
TypeError: attach() missing 1 required positional argument: 's2'
```

Guess The Output

```
def grocery(name,price):
    print("Item is",name,"It's price is",price)

grocery("Bread",20)
grocery(150,"Butter")
```

Output:



```
Item is Bread It's price is 20
Item is 150 It's price is Butter
```

The Problem With Positional Arguments

- The problem with **positional arguments** is that they always **bind** to the **position** of parameters.

- So **1st argument** will be copied to **1st parameter** , **2nd argument** will be copied to **2nd parameter** and so on.

- Due to this in the previous example the value **150** was copied to **name** and "**Butter**" was copied to **price**

- To solve this problem , **Python** provides us the concept of **keyword arguments**

Keyword Arguments

- **Keyword arguments** are arguments that identify parameters with their names

- With **keyword arguments** in **Python**, we can change the order of passing the arguments without any consequences

- **Syntax:**

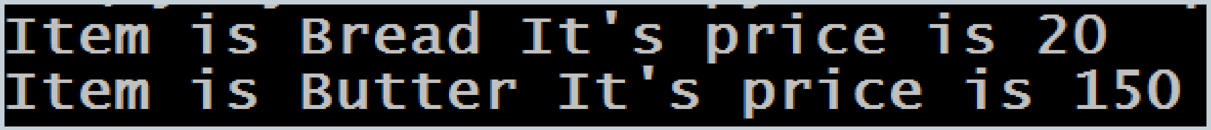
function_name(paramname1=value,paramname2=value)

Complete Example

```
def grocery(name,price):
   print("Item is",name,"It's price is",price)

grocery(name="Bread",price=20)
grocery(price=150,name="Butter")
```

Output:



```
Item is Bread It's price is 20
Item is Butter It's price is 150
```

Point To Remember!

- A **positional argument** can never follow a **keyword argument** i.e. the keyword argument should always appear after positional argument

- **For example:**
  - **def display(num1,num2):**
    - **# some code**

Now if we call the above function as:

display(10,num2=15)

Then it will be correct. But if we call it as:

display(num1=10,15)

Then it will be a **Syntax Error**

Default Arguments

- For some functions, we may want to make some parameters ***optional*** and use **default values** in case the user does not want to provide values for them.

- This is done with the help of **default argument** values.

- We can specify **default argument** values for parameters by appending to the parameter name in the function definition the assignment operator (=) followed by the **default value**.

- **Syntax:**

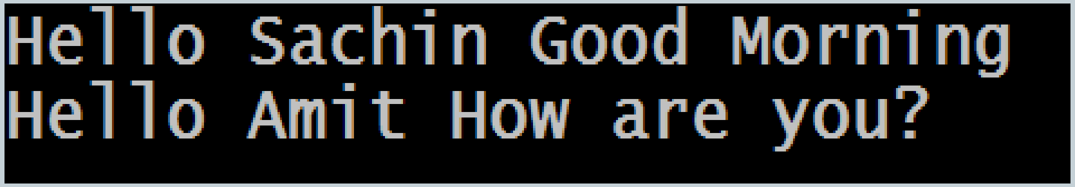
```
def  function_name(paramname1=value,paramname2=value):
    # function body
```

Complete Example

```
def greet(name,msg="Good Morning"):
    print("Hello",name,msg)

greet("Sachin")
greet("Amit","How are you?")
```

Output:



```
Hello Sachin Good Morning
Hello Amit How are you?
```

Point To Remember!

- A function can have any **number of default arguments** but t once we have a default argument, all the arguments to **it's right must also have default values**.

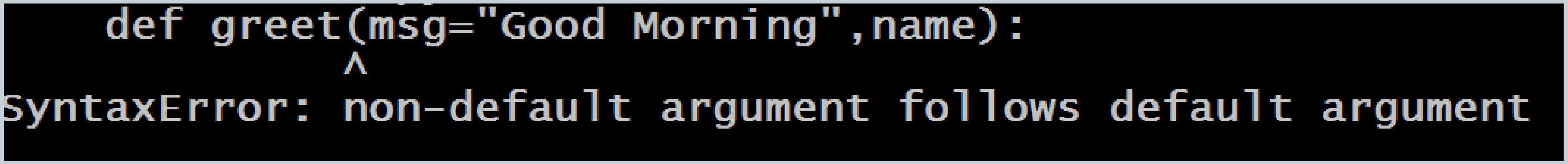
- This means to say, **non-default arguments** cannot follow **default arguments**.

Point To Remember!

- **For example:** if we had defined the function header above as:

def greet(msg = "Good morning!", name):

- **Then we would have got the following SyntaxError**



```
    def greet(msg="Good Morning",name):
             ^
SyntaxError: non-default argument follows default argument
```

Point To Remember!

- If a function has **default arguments** , set then while calling it if we are **skipping** an argument then **we must skip all the arguments after it also**.
- **For example:**

```
def show(a=10,b=20,c=30):
    print(a,b,c)
```

- Now , if we call the above function as :

show(5)

- It will work and output will be **5 20 30**

- If we call it as :

show(5,7)

- Still it will work and output will be **5 7 30**
- But if we call it as

show(5, ,7)

- Then it will be an error

The solution to this problem is to use **default argument** as **keyword argument** :

show(5,c=7)

This will give the output as

5 20 7

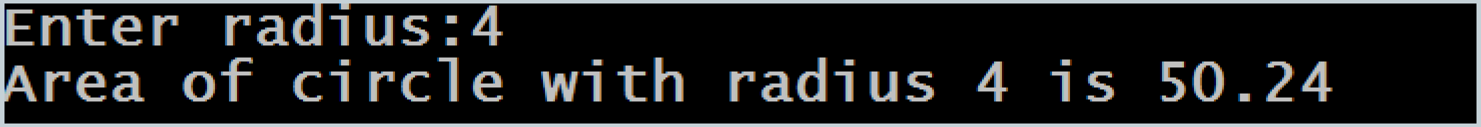
Exercise

- Write a function called **cal_area( )** using **default argument** concept which accepts **radius** and **pi** as arguments and calculates and displays area of the Circle. The value of **pi** should be used as **default argument** and value of **radius** should be **accepted from the user**

Solution

```python
def cal_area(radius,pi=3.14):
    area=pi*radius**2
    print("Area of circle with radius",radius,"is",area)

rad=int(input("Enter radius:"))
cal_area(rad)
```



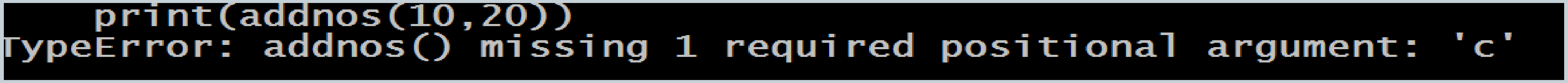
```
Enter radius:4
Area of circle with radius 4 is 50.24
```

Guess The Output ?

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

def addnos(a,b,c):
   d=a+b+c
   return d
print(addnos(10,20))
print(addnos(10,20,30))
```

Output:



```
    print(addnos(10,20))
TypeError: addnos() missing 1 required positional argument: 'c'
```

Why Did The Error Occur ?

- The error occurred because **Python** does not support **Function** or **Method Overloading**
- **Moreover Python understands the latest definition of a function addnos( ) which takes 3 arguments**
- Now since we passed **2 arguments** only , the call generated **error** because Python tried to call the method with **3 arguments**

Solution

- The solution to this problem is a technique called **variable length arguments**

- In this technique , we define the function in such a way that it can accept any number of arguments from **0** to **infinite**

Syntax Of
Variable Length Arguments

def <function_name>(* <arg_name>):

Function body

- As we can observe , to create a function with **variable length arguments** we simply prefix the argument name with an **asterisk**.
- **For example:**

def addnos(*a):

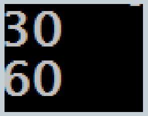
- The function **addnos()** can now be called with as many **number of arguments** as we want and all the arguments will be stored inside the argument **a** which will be internally treated as **tuple**

Complete Example

```
def addnos(*a):
   sum =0
   for x in a:
         sum=sum+x
   return sum

print(addnos(10,20))
print(addnos(10,20,30))
```

Output:



```
30
60
```

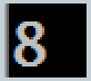
Exercise

- Write a function called **find_largest( )** which accepts multiple strings as argument and returns the length of the largest string

Solution

```
def findlargest(*names):
   max=0
   for s in names:
         if len(s)>max:
                  max=len(s)
   return max
print(findlargest("January","February","March"))
```

Output:



```
8
```

Exercise

- Modify the previous example so that the function **find_largest( )** now returns the largest string itself and not it's length

Solution

```
def findlargest(*names):
   max=0
   largest=""
   for s in names:
         if len(s)>max:
                 max=len(s)
                 largest=s
   return largest
print(findlargest("January","February","March"))
```

Output:

```
February
```

Point To Remember!

- A function cannot have 2 variable length arguments. So the following is wrong:

```
def addnos(*a,*b):
```

Point To Remember!

- If we have any other argument along with **variable length argument** , then it should be set **before** the **variable length argument**

```
def addnos(n,*a):
    sum =n
    for x in a:
            sum=sum+x
    return sum
print(addnos(10,20,30))
print(addnos(10,20,30,40))
```

Point To Remember!

- If we set the other argument used with **variable length argument** , **after** the **variable length argument then while calling it , we must pass it as keyword argument**

```
def addnos(*a,n):
   sum =n
   for x in a:
          sum=sum+x
   return sum
print(addnos(20,30,n=10))
print(addnos(20,30,40,n=10))
```

Guess The Output

```
def addnos(*a,n):
   sum =n
   for x in a:
         sum=sum+x
   return sum
print(addnos(20,n=10,30))
```

Output:

```
SyntaxError: Positional argument follows keyword argument
```

Guess The Output

```
def show(a,b,c=3,d=4):
    print(a,b,c,d)

show(10,20)
```

Output:

```
10 20 3 4
```

Guess The Output

```
def show(a,b,c=3,d=4):
    print(a,b,c,d)

show(10,20,30,40)
```

Output:

```
10 20 30 40
```

Guess The Output

```
def show(a,b,c=3,d=4):
   print(a,b,c,d)

show(d=10,a=20,b=30)
```

Output:

```
20 30 3 10
```

Guess The Output

```
def show(a,b,c=3,d=4):
    print(a,b,c,d)

show()
```

Output:

TypeError

Guess The Output

```
def show(a,b,c=3,d=4):
    print(a,b,c,d)

show(c=30,d=40,10,20)
```

Output:

```
SyntaxError
```

Guess The Output

```
def show(a,b,c=3,d=4):
   print(a,b,c,d)

show(30,40,b=15)
```

Output:

```
TypeError : got multiple values for argument ‘b’
```