More on User Defined Functions

February 7, 2023

[1]:
```
def location(city, country, continent):
    print(f'{city} is in {country} which is in {continent}')

location('Mumbai', 'India', 'Asia')
```

```
Mumbai is in India which is in Asia
```

[2]:
```
def location(city, country, continent):
    print(f'{city} is in {country} which is in {continent}')

location(continent='Asia', city='Mumbai', country='India')
```

```
Mumbai is in India which is in Asia
```

[ ]:
```
lst = [1, -7,  13, 12]
lst.sort() # ascending order of elements
print(lst)
```

[3]:
```
lst = [1, -7,  13, 12]
lst.sort() # ascending order of elements
print(lst)
```

```
[-7, 1, 12, 13]
```

[ ]:
```
reverse, key
reverse = False
key = None
```

[4]:
```
lst = [1, -7,  13, 12]
lst.sort(reverse = True) # ascending order of elements
print(lst)
```

```
[13, 12, 1, -7]
```

[5]:
```
names = ['rocky', 'garuda', 'adheera', 'tina']
# ['adheera', 'garuda', 'rocky', 'tina']
names.sort() # reverse = False, key = fun
print(names)
```

```
['adheera', 'garuda', 'rocky', 'tina']
```

[6]:
```python
names = ['rocky', 'garuda', 'adheera', 'tina']
# ['tina', 'rocky', 'garuda', 'adheera']
names.sort(key = len) # reverse = False, key = fun
print(names)
```

```
['tina', 'rocky', 'garuda', 'adheera']
```

[7]:
```python
def location(city, country, continent='Asia'):
    # 2 positional arguments city, country --> must
    # 1 default argument
    print(f'{city} is in {country} which is in {continent}')

location('Mumbai', 'India')
```

```
Mumbai is in India which is in Asia
```

[8]:
```python
def location(city, country, continent='Asia'):
    # 2 positional arguments city, country --> must
    # 1 default argument
    print(f'{city} is in {country} which is in {continent}')

location('Shaghai', 'China')
```

```
Shaghai is in China which is in Asia
```

[9]:
```python
def location(city, country, continent='Asia'):
    # 2 positional arguments city, country --> must
    # 1 default argument
    print(f'{city} is in {country} which is in {continent}')

location('Berlin', 'Germany', 'Europe')
```

```
Berlin is in Germany which is in Asia
```

[10]:
```python
# After default parameters we cannot write positional parameters
def location(city, country='India', continent):
    # 2 positional arguments city, country --> must
    # 1 default argument
    print(f'{city} is in {country} which is in {continent}')

location('Berlin', 'Germany', 'Europe')
```

```
  File "C:\Users\Unstoppable Force\AppData\Local\Temp\ipykernel_26720\295087946
↪py", line 2
    def location(city, country='India', continent):
                                                  ^
SyntaxError: non-default argument follows default argument
```

```
[11]: # After default parameters we cannot write positional parameters
def location(city='Mumbai', country='India', continent):
    # 2 positional arguments city, country --> must
    # 1 default argument
    print(f'{city} is in {country} which is in {continent}')

location('Berlin', 'Germany', 'Europe')
```

```
  File "C:\Users\Unstoppable Force\AppData\Local\Temp\ipykernel_26720\249790968
↪py", line 2
    def location(city='Mumbai', country='India', continent):
                                                 ^
SyntaxError: non-default argument follows default argument
```

```
[12]: # After default parameters we cannot write positional parameters
def location(city='Mumbai', country, continent):
    # 2 positional arguments city, country --> must
    # 1 default argument
    print(f'{city} is in {country} which is in {continent}')

location('Berlin', 'Germany', 'Europe')
```

```
  File "C:\Users\Unstoppable Force\AppData\Local\Temp\ipykernel_26720\613751054
↪py", line 2
    def location(city='Mumbai', country, continent):
                                ^
SyntaxError: non-default argument follows default argument
```

```
[16]: # After default parameters we cannot write positional parameters
def location(city='Mumbai', country='India', continent='Asia'):
    # 3 default arguments
    print(f'{city} is in {country} which is in {continent}')

location() # without arguments
location('Hyderabad') # 1 argument
location('Karachi', 'Pakistan') # 2 arguments
location('Nairobi', 'Kenya', 'Africa')
```

```
Mumbai is in India which is in Asia
Hyderabad is in India which is in Asia
Karachi is in Pakistan which is in Asia
Nairobi is in Kenya which is in Africa
```

1 Doc Strings in functions

[18]:
```
print(sum.__doc__)
```

```
Return the sum of a 'start' value (default: 0) plus an iterable of numbers

When the iterable is empty, return the start value.
This function is intended specifically for use with numeric values and may
reject non-numeric types.
```

[20]:
```
help(sum)
```

```
Help on built-in function sum in module builtins:

sum(iterable, /, start=0)
    Return the sum of a 'start' value (default: 0) plus an iterable of numbers

    When the iterable is empty, return the start value.
    This function is intended specifically for use with numeric values and may
    reject non-numeric types.
```

[21]:
```
def add(a, b):
    """
    Accepts two integers a and b
    Returns a + b
    """
    return a + b

print(add(10, 20))
```

```
30
```

[22]:
```
print(add.__doc__)
```

```

    Accepts two integers a and b
    Returns a + b

```

[23]:
```
help(add)
```

```
Help on function add in module __main__:

add(a, b)
    Accepts two integers a and b
    Returns a + b
```

[24]:
```
print(10, 20, 30, 40) # ends with a new line
print('Hello')
```

```
10 20 30 40
Hello
```

[25]:
```
print(10, 20, 30, 40, ',', 'pavan') # ends with a new line
print('Hello')
```

```
10,20,30,40pavanHello
```

[26]:
```
# After default parameters we cannot write positional parameters
def location(city, country, continent='Asia'):
    # 2 positional arguments city, country --> must
    # 1 default argument
    print(f'{city} is in {country} which is in {continent}')

location('Tokyo', 'Japan', 'South America')
```

```
Tokyo is in Japan which is in South America
```

[27]:
```
# After default parameters we cannot write positional parameters
def location(city, country, continent='Asia'):
    # 2 positional arguments city, country --> must
    # 1 default argument
    print(f'{city} is in {country} which is in {continent}')

location('Tokyo', 'Japan', 'South America')
```

```
Tokyo is in Japan which is in South America
```

[28]:
```
max(1, 2)
```

[28]:
```
2
```

[29]:
```
def my_max(a, b):
    return a if a > b else b

my_max(1, 2)
```

[29]:
```
2
```

[30]:
```
max(1, 2, 3)
```

[30]:
```
3
```

[31]:
```
my_max(1, 2, 3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
~\AppData\Local\Temp\ipykernel_26720\2521420054.py in <cell line: 1>()
----> 1 my_max(1, 2, 3)

TypeError: my_max() takes 2 positional arguments but 3 were given
```

[32]:
```
max(1, 2, 3, 4)
```

[32]:
```
4
```

2 The creation of a function which takes arbitrary number of arguments

[41]:
```
def Sum(*A):
    s = 0
    for i in A:
        s += i
    return s

Sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Sum(1, 2, 3, 4, 5, 6, 7, 8, 9)
Sum(1, 2, 3, 4, 5, 6, 7, 8)
```

[41]:
```
36
```

[33]:
```
a, b = 10, 20
print(a, b)
```

```
10 20
```

[34]:
```
a, b, c = 10, 20, 30
print(a, b, c)
```

```
10 20 30
```

[36]:
```
a = 10, 20, 30
print(a)
print(type(a))
```

```
(10, 20, 30)
<class 'tuple'>
```

[43]:
```
def Sum(*A, s=0):
    for i in A:
        s += i
    return s

print(Sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, s=100))
print(Sum(1, 2, 3, 4, 5, 6, 7, 8, 9))
print(Sum(1, 2, 3, 4, 5, 6, 7, 8))
```

```
155
45
36
```

[45]:
```
print(10, 20, 30, sep=',')
```

```
10,20,30
```

[48]:
```
sum([10, 20, 30], 10)
```

[48]:
```
70
```

[49]:
```
# Create a function which takes no arguments but returns something
def get_i():
    return int(input())

def get_f():
    return float(input())

def get_s():
    return str(input())


a = get_i()
b = get_i()
c = get_i()
print(a + b + c)
```

```
10
20
30
60
```

[52]:
```
def arithmetic_operations(a, b):
    return (a + b, a - b, a * b, a // b)

res = arithmetic_operations(10, 3)
print(res)
print(type(res))
```

```
(13, 7, 30, 3)
<class 'tuple'>
```

[53]:
```
# Generic read() function
def read(datatype, single=0):
    if single == 0:
        return map(datatype, input().split())
    else:
        return datatype(input()) # int(input())

n = read(int, 1)
print(n * n)
```

```
5
25
```

[ ]:
```
# int --> int(input())
# float --> float(input())
# str --> str(input())

# map(int, input().split())
# map(float, input().split())
# map(str, input().split())
```

[54]:
```
# Generic read() function
def read(datatype, single=0):
    if single == 0:
        return map(datatype, input().split())
    else:
        return datatype(input()) # int(input())

a, b, c = read(int)
print(a + b + c)
```

```
10 20 30
60
```

[55]:
```
# Generic read() function
def read(datatype, single=0):
    if single == 0:
        return map(datatype, input().split())
    else:
        return datatype(input()) # int(input())

lst = list(read(int))
print(sum(lst))
```

```
1 2 3 4
10
```