PYTHON

From Simple to Complex
With
Examples

AYESHA NOREEN

Bachelor's in Software Engineering,

Master's in Computer Science

from COMSATS University, Islamabad

NOTE!!!

In these notes Screenshots of practice examples and coding are added. The code files are also available in code folder that contain .ipynb files that are created on Jupyter notebook.

Chapter14
*args and **kwargs

If we want a function that will return the sum of its parameters and number of parameters are not known.

i.e. def sum(a,b):

return a+b

print(sum(1,1,4))

It gives error because sum function that we define will only accept two parameters but we are passing three. To solve this problem we use *operator *args. It return a tuple.

A program that print sum of its parameters

```
def sum(*args):   #here we can pasa any name e.g *numbers but args mean arguments is best
    print(args)  #it contains a tuple of all values that we pass in function call
    print(type(args))  #its type is <class  'tuple'>
    total=0
    for i in args:
        total+=i
    return total
print(sum(1,2,3,4,5))   #it prints 15 we can take many values according to our need
```

```
(1, 2, 3, 4, 5)
<class 'tuple'>
15
```

- *args with normal parameters

```
def multiply(num1,num2,*args):
    mul=1
    for i in args:
        mul*=i
    return mul
print(multiply(1,2,6,2,1))  #it prints 12=6*2*1
print(multiply(3,4))#it return 1 bcz 3 is num1 and 4 is num2
#print(multiply()) #error because we must have to pass 2 or 3 arguments
```

```
12
1
```

- *args as argument

```
def multiply(*args):
    mul=1
    for i in args:
        mul*=i
    return mul
numbers=[1,3,1,6,2]
print(multiply(numbers))   #it prints [1, 3, 1, 6, 2] multiply function does not work
print(*numbers)    # it prints 1 3 1 6 2 mean by use of * list becomes unpack
print(multiply(*numbers)) #it prints 36 bcz now list is unpack and multiply function works
```

```
[1, 3, 1, 6, 2]
1 3 1 6 2
36
```

- **TODO Task**

Define a function that takes a number and args as parameters and print each element of list raised to the power of number

```
#define a function that takes a nummber and args as paramters and print each element of
#list raised to the power of number
def power(num,*args):
    if args:
        return [i**num for i in args]
    else:
        return "you did not pass any args"
print(power(2,*[1,2,3]))
#it prints[1,4,9]
```

```
[1, 4, 9]
```

- *kwargs (Keyword argument)

```
def fun(**kwarg):
    print(kwarg)
d={
    'name':'ayesha',
    'age':22
}
fun(**d)
```

```
{'name': 'ayesha', 'age': 22}
```

- **TODO Task**

Define a function which take a name and a dictionary of names than print it.

```
def fun(name,**kwarg):
    for  k,v in kwarg.items():
        print(f"key is={k}:value is={v}")
names={'name1':'ayesha','name2':'noreen','name3':'sana'}
fun('Adnan',**names)
```

```
key is=name1:value is=ayesha
key is=name2:value is=noreen
key is=name3:value is=sana
```

- ## A function with all type of parameters

If we have a function with all type of parameters than we have to follow this order. As,
(Normal parameters, *args, default parameters, **kwargs)

```
# Function with all type of parameters
def fun(name,*args,dname1='unknown',dname2='none',**kwarg):
    print(f"normal parameter is={name}")
    print(f"*args is={args}")
    print(f"Default parameter is={dname1} and {dname2}")
    print(f"**kwarg is={kwarg}")
print(fun('ayesha',1,2,3,first_name='Ayesha',last_name='Noreen',age=22))
```

```
normal parameter is=ayesha
*args is=(1, 2, 3)
Default parameter is=unknown and none
**kwarg is={'first_name': 'Ayesha', 'last_name': 'Noreen', 'age': 22}
None
```

- **TODO Task**

Make a function that take either 1 or 2 arguments 1st argument is a list and 2nd argument is reverse_str=True if only 1 argument take than print list with 1st letter capital and if 2 arguments than print both list with 1st letter capital and reverse string with 1st letter capital.

```
def fun(l,**kwarg):
    if kwarg.get('reverse_str')==True:
        return[i[::-1].title() for i in l]
    else:
        return [name.title() for name in l]
names=['ayesha','noreen','sana','rehman']
print(fun(names))
print(fun(names,reverse_str=True))
```

```
['Ayesha', 'Noreen', 'Sana', 'Rehman']
['Ahseya', 'Neeron', 'Anas', 'Namher']
```