Functions

- When the function is called, the arguments that are passed are bound to the parameters in order
- **How to pass the data into a function?**
  - **Required arguments**
  - **Keyword Arguments**

Preview Exercise:

What will be the output of the following Python code?

```
def func(a, b):
    print('a is', a, 'and b is', b)

c=15
d=20
func(3, 7)
func(a=25, b= 24)
func(c, d)
```

Functions as Arguments

- **Required arguments**

When the function is called, specify a corresponding list of arguments as defined in the function.

- **You can directly pass the data as arguments**

```
def hello_func(name, somelist):
    print("Hello,", name)
    return 1, 2

a,b = hello_func("Ben",[1,2])

print(a, b)
```

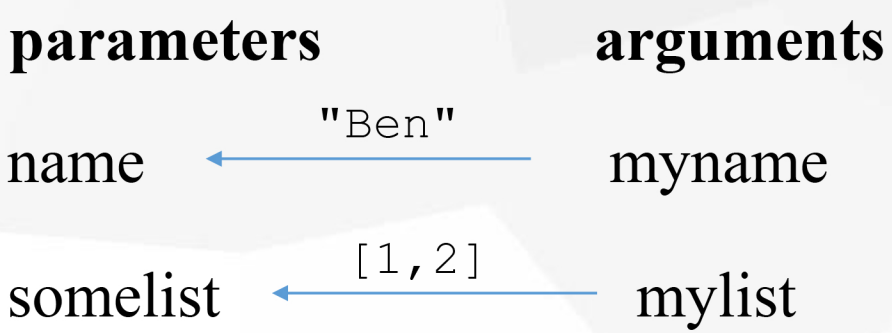
Hello, Ben
1 2

Functions as Arguments

- **Required arguments**

When the function is called, specify a corresponding list of arguments as defined in the function.

- **Or pass the data by reference.**




```
def hello_func(name, somelist):
    print("Hello,", name)
    return 1, 2

myname = "Ben"
mylist = [1,2]
print(myname, mylist)
a,b = hello_func(myname, mylist)

print(a, b)
```

Ben [1, 2]
Hello, Ben
1 2

Functions as Arguments

- **Keyword arguments**
- specify arguments in the form <keyword>=<value>.
- In this case, each <keyword> must match a parameter in the Python function definition

```
def hello_func(name, somelist):
    print("Hello,", name)
    return 1, 2

a,b = hello_func(name= "Ben",
somelist= [1,2])

print(a, b)
```

```
Hello, Ben
1 2
```

Functions as Arguments

- Parameters can be passed by **reference**.
- However, **only mutable objects (like list, tuple)** can be changed in the called function.

somelist is changed in the called function

```
def hello_func(name, somelist):
    print("Hello,", name)
    somelist[0] = 3
    return 1, 2

myname = "Ben"
mylist = [1,2]
print(myname, mylist)
a,b = hello_func(myname, mylist)

print(mylist)
print(a, b)
```

Ben [1, 2]
Hello, Ben
[3, 2]
1 2

Functions as Arguments

- Be careful, **immutable** objects can not be changed in the called function.

```
def hello_func(name, somelist):
    print("Hello,", name)
    name= "John"
    return 1, 2

myname = "Ben"
mylist = [1,2]
print(myname, mylist)
a,b = hello_func(myname, mylist)

print(myname)
print(a, b)
```

- myname is a string, and can not be changed in the called function
- name= "John" become a local variable in the function

Ben [1, 2]
Hello, Ben
Ben
1 2

单选题 1分

设置

What will be the output of the following Python code?

```
def power(x, y=2):
    r = 1
    for i in range(y):
        r = r * x
    return r
print (power(3))
print (power(3, 3))
```

A
212
32

B
9
27

C
567
98

D
None of the mentioned

提交

主观题 10分

设置

```
def add (x, y):
    z= x+ y
    print(z)
    return z

x = 1
y = 2
print(add(x,y))
```

What are the results?

正常使用主观题需2.0以上版本雨课堂

作答

Functions as Arguments

- ***Arbitrary Arguments***: If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- This way the function will receive a *tuple* of arguments, and can access the items accordingly:

```
def my_function(*kids):
    print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

```
The youngest child is Linus
```

设置

- Arguments can even be functions

```
def func_a():
    print ('inside func_a')
def func_b(y):
    print('inside func_b')
    return y
def func_c(z):
    print ('inside func_c')
    return z

print (func_a())
print (5 + func_b(2))
print(func_c(func_b(3)))
```

课堂

作答

Functions as Arguments

- Arguments can even be functions

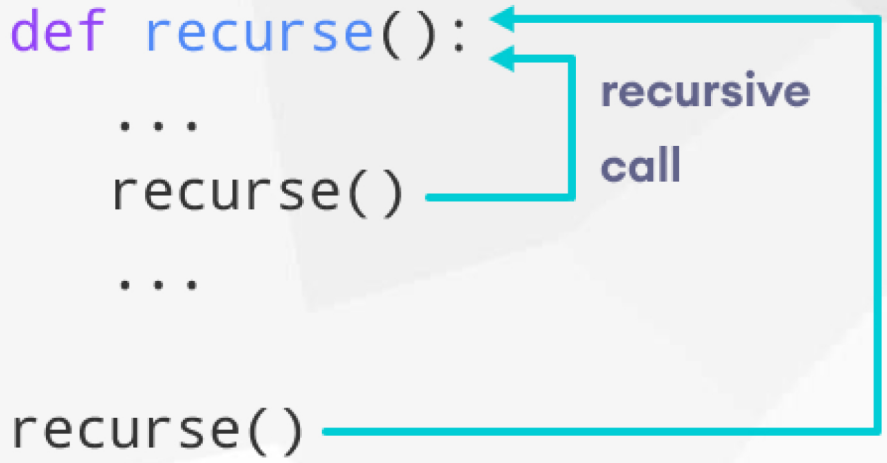
```
def func_a():
    print ('inside func_a')
def func_b(y):
    print('inside func_b')
    return y
def func_c(z):
    print ('inside func_c')
    return z

print (func_a())
print (5 + func_b(2))
print(func_c(func_b(3)))
```

```
inside func_a
None
inside func_b
7
inside func_b
inside func_c
3
```

Functions as Arguments

- ***function recursion:*** we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.



Functions as Arguments

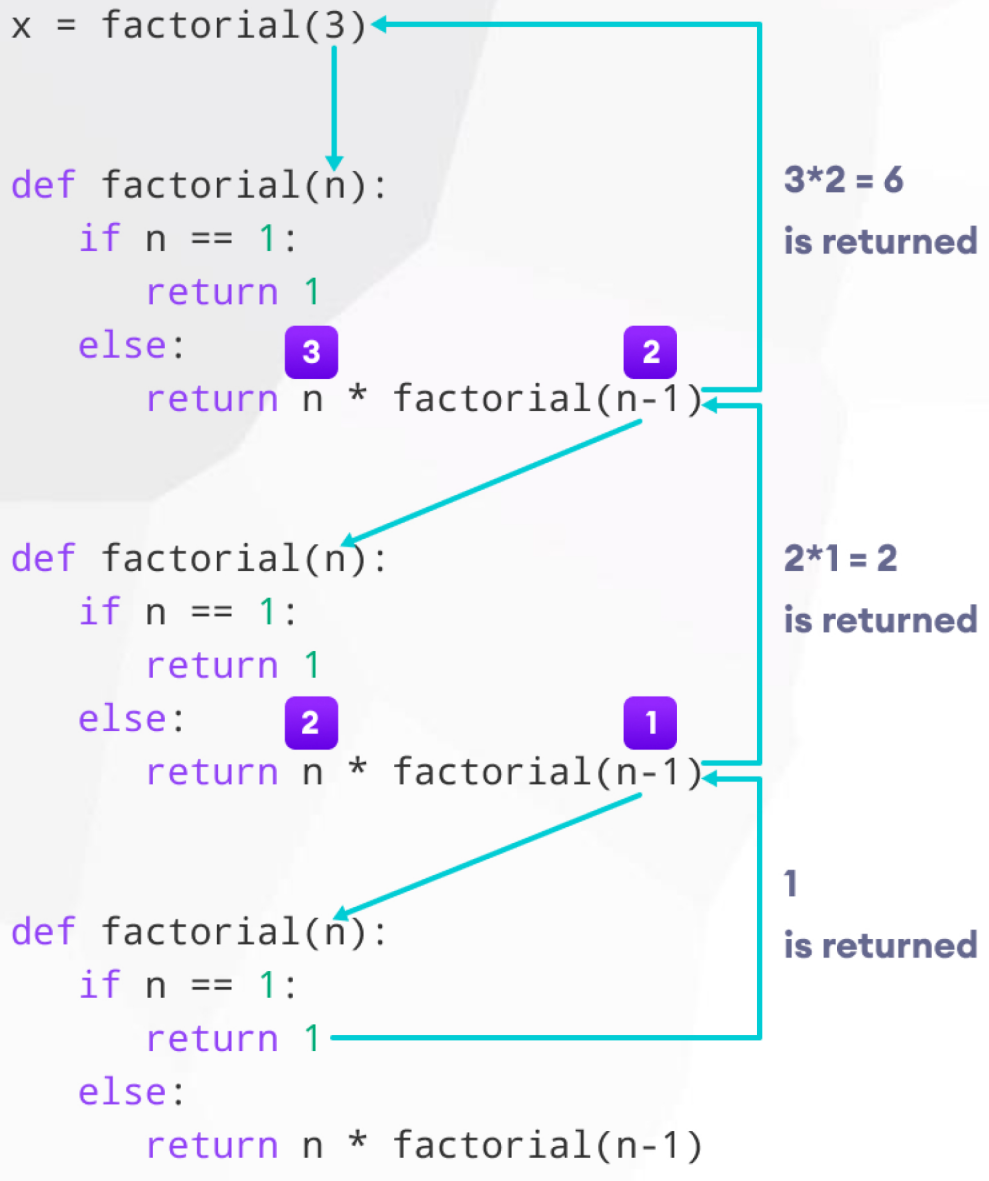
An example of a recursive function to find the factorial of an integer

```
def factorial(x):
    "find the factorial of an integer"
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))
num = 3
print("The factorial of", num, "is", factorial(num))
```

Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is 1*2*3*4*5*6 = 720.

Functions as Arguments

```
# 1st call with 3
factorial(3)
# 2nd call with 2
3 * factorial(2)
# 3rd call with 1
3 * 2 * factorial(1)
# return from 3rd call as number=1
3 * 2 * 1
# return from 2nd call
3 * 2
# return from 1st call
6
```



主观题 10分

设置

- *Could you try to print out the result of this recursion?*

```
def tri_recursion(k):
    if(k>0):
            result = k+tri_recursion(k-1)
            print(result)
    else:
            result = 0
    return result
print("\n\nRecursion Example Results")
tri_recursion(6)
```

```
Recursion Example Results
1
3
6
10
15
21
```