Functions

Blocks

A set of instruction which work together

We saw blocks in while loop

```
number = 1
lastnumber = 0
while number < 200:
      print (number)
      number = number * 2
      number =number + 1
      lastnumber = number
```

Functions

1- A function is a block with a name

2- A function is a piece of code that is called by name. It can be passed data to operate on (ie. the parameters) and can optionally return data (the return value).

```
def name (arg1, arg2, ...):
    statement1
    statement2
    statement3
    ……….
    return expression
```

Functions

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Functions

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).

def *name* (*arg1*, *arg2*, ...):

"""*documentation*""" # optional doc string

statements

return *expression* # from function

Functions

- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.

```
def name (arg1, arg2, ...):
    """documentation"""    # optional doc string
    statements

    return expression      # from function
```

Functions

- The code block within every function starts with a colon (:) and is indented.

```
def name (arg1, arg2, ...):
    """documentation"""    # optional doc string
    statements

    return expression      # from function
```

Functions

- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.

```
def name (arg1, arg2, ...):
    """documentation"""   # optional doc string
    statements

    return expression     # from function
```

Functions

The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

```
def name (arg1, arg2, ...):
    """documentation"""   # optional doc string
    statements

    return expression     # from function
```

Functions

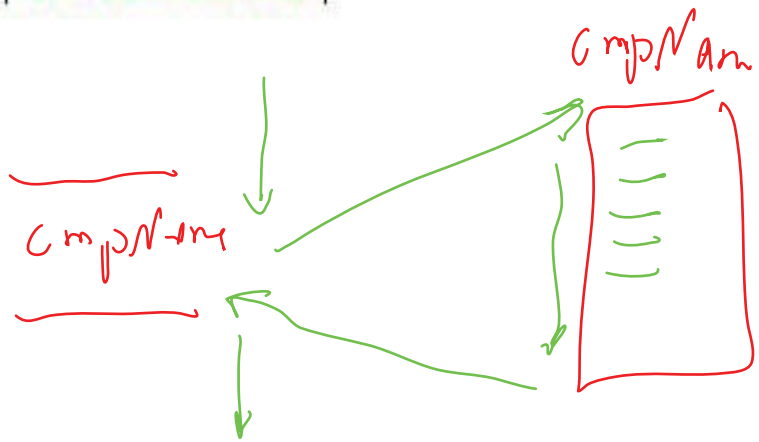
```
def name (arg1, arg2, ...):
    """documentation"""   # optional doc string
    statements

    return expression     # from function
```

Functions

fimple-fun.py - C:/Documents and Settings/admin/Desktop/intro-python/exam...

File Edit Format Run Options Window Help

```
def cmpName():
    Astr = input("Enter first name:")
    Bstr = input("Enter second name:")
    if (Astr!=Bstr):
        print("These names are not equal")
    else:
        print("These names are  equal")

print("The function start")
cmpName()
print("The function end")
```



Python 3.4.4 Shell

File Edit Shell Debug Options

```
Python 3.4.4 (v3.4.4:737efc
tel)] on win32
Type "copyright", "credits"
>>>
 RESTART: C:/Documents and
ple-fun.py
The function start
Enter first name:jim
Enter second name:alen
These names are not equal
The function end
>>>
```

Functions

1- A function is a block with a name

2- A function is a piece of code that is called by name. It can be passed data to operate on (ie. the parameters) and can optionally return data (the return value).

```
def name (arg1, arg2, ...):
    """documentation"""   # optional doc string
    statements

    return expression
```

Functions

```
def name (arg1, arg2, ...):
    """documentation"""   # optional doc string
    statements
```

Formal

```
name(actual_arg1, actual_arg2, ….)
```

actual

Functions

```
def name (arg1, arg2, ...):
    """documentation"""   # optional doc string
    statements

    return expression        # from function


var=name(actual_arg1, actual_arg2, ….)
```

Functions

```
def name (arg1, arg2, ...):
    """documentation"""   # optional doc string
    statements

    return expression        # from function


name(actual_arg1, actual_arg2, ….)
```

Functions

fimple-fun.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/9a-1/fimple-fun.py (3.4.4)

File Edit Format Run Options Window Help

```
# Function definition is here
def printme( tstr, a ):
   "This prints a passed string into this function"
   print (tstr)
   print (a)
   return;

# Now you can call printme function
printme("I'm first call to user defined function!", 12)
c=17
printme("Again second call to the same function",c)
```

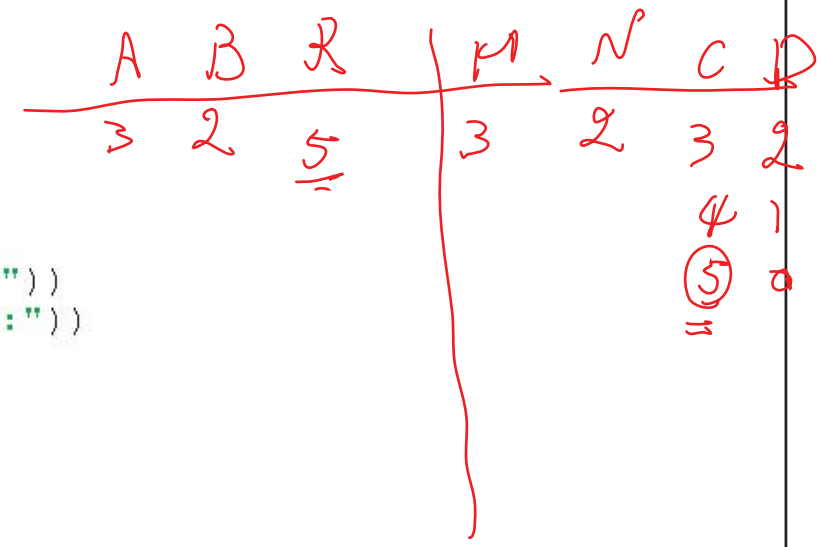
```
I'm first call to user defined function!
12
Again second call to the same function
17
>>>
```

Functions

fuadd.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\9a-1\fuadd.py (3.4.4)

File Edit Format Run Options Window Help

```
def add(M, N):
    C=M
    D=N
    while (D!=0) :
        D=D-1
        C=C+1
    return (C)
A = int(input("Enter first number:"))
B = int(input("Enter second number:"))
R = add(A, B)
print("The summation is: ", R)
```




```
Enter first number:23
Enter second number:12
The summation is:  35
>>>
```

Scope of variables

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python –

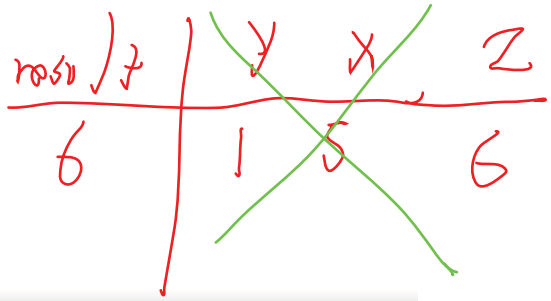
- Global variables
- Local variables

Scope of variables

local.py - D:/nowzari/awork/course/course/python/python-my/python/examples/09-functions/local.py (3.4.4)

File Edit Format Run Options Window Help

```
def func(Y):            # Y and Z assigned in function: locals
    # Local scope
    X=5
    Z = X + Y           # X is a local
    return Z


result=func(1)
print("The result is: ", result)
```



Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

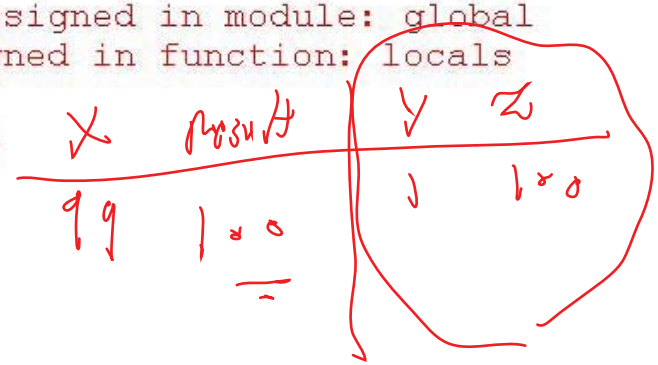
```
Python 3.4.4 (v3.4.4:737efcadf5a6,
tel)] on win32
Type "copyright", "credits" or "lic
>>>
 RESTART: D:/nowzari/awork/course/c
ctions/local.py
The result is:  6
>>>
```

Scope of variables

glob.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/9b-1/glob.py (3.4.4)

File Edit Format Run Options Window Help

```
# Global scope
X = 99                  # X and func assigned in module: global
def func(Y):            # Y and Z assigned in function: locals
    # Local scope
    Z = X + Y           # X is a global
    return Z


result=func(1)
print("The result is: ", result)
```



Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

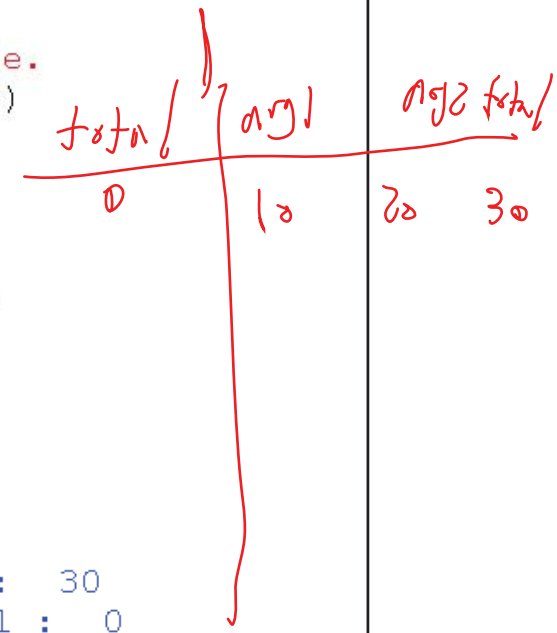
```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2C
tel)] on win32
Type "copyright", "credits" or "license()" f
>>>
 RESTART: C:/Documents and Settings/admin/De
b.py
The result is:  100
>>>
```

Scope of variables

fuadd.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\9a-1\fuadd.py (3.4.4)

File Edit Format Run Options Window Help

```
total = 0; # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
   total = arg1 + arg2; # Here total is local variable.
   print ("Inside the function local total : ", total)
   return total;

# Now you can call sum function
sum( 10, 20 );
print ("Outside the function global total : ", total)
```



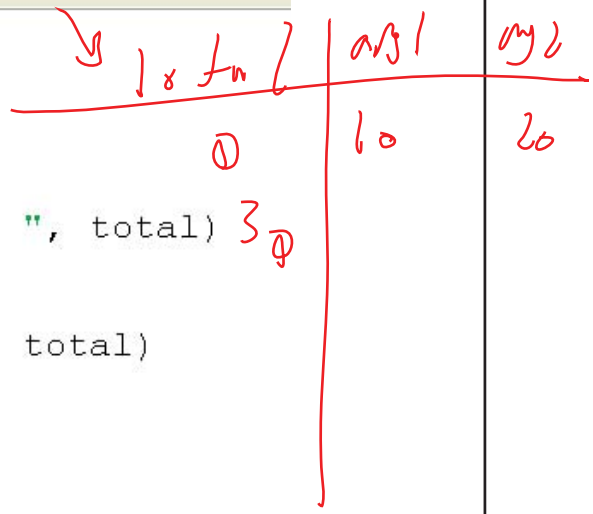

```
Inside the function local total :  30
Outside the function global total :  0
>>>
```

Scope of variables

glob1.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/9b-1/glob1.py (3.4.4)

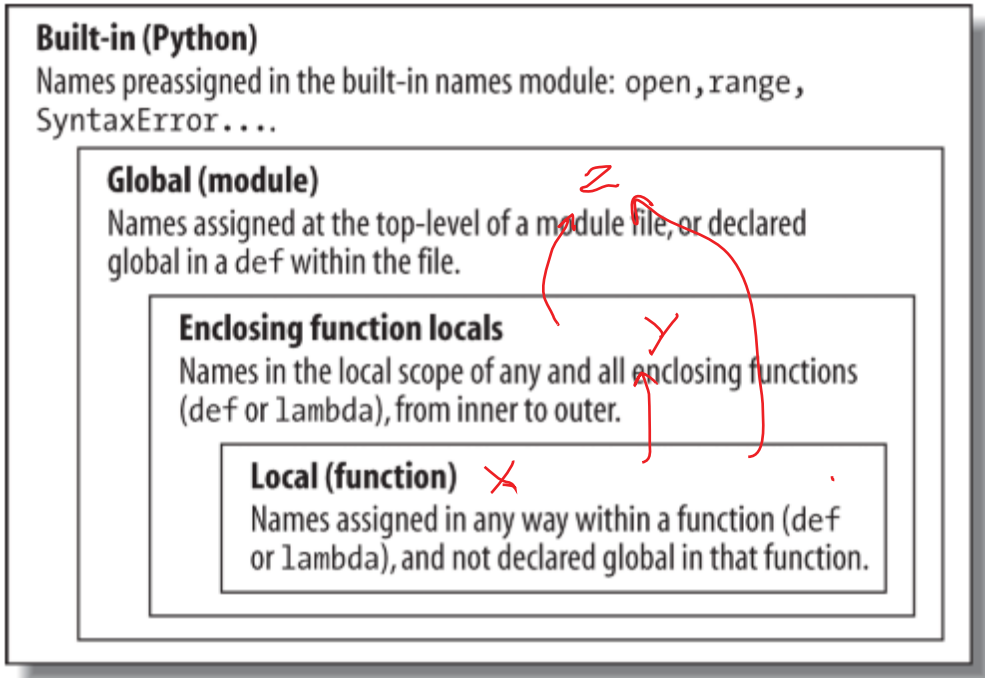
File Edit Format Run Options Window Help

```
total=0

def sum(arg1, arg2):
    global total
    total = arg1 + arg2
    print("Inside the function global total : ", total)

sum(10, 20)
print("Outside the function global total : ", total)
```




```
Inside the function global total :  30
Outside the function global total :  30
```

Scope of variables



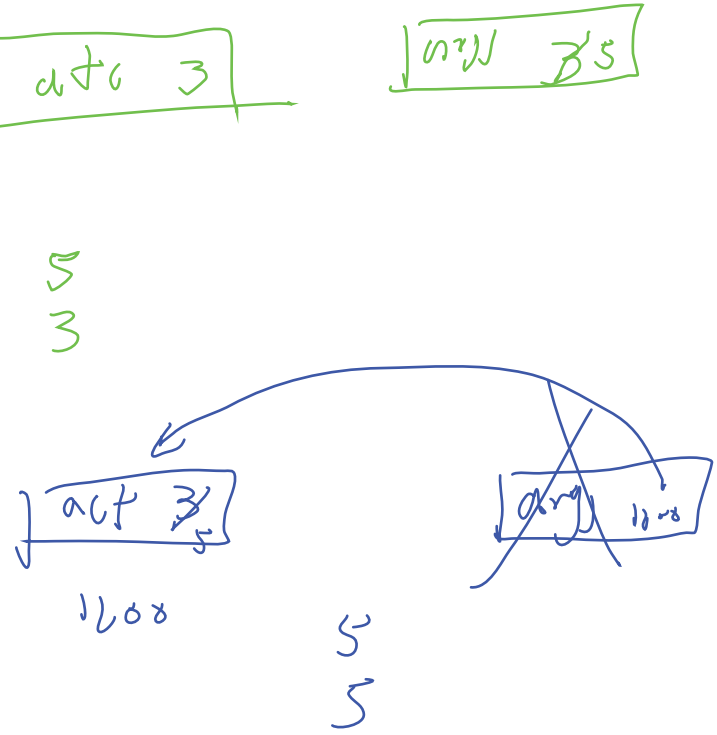
Parameter passing

1- pass by value

2- pass by reference

```
f ( arg1 ):
    arg1 = 5
    print ( arg1 )

act = 3
f ( act )
print ( act )
```

5
3



Parameter passing

If you pass immutable arguments like integers, strings or tuples to a function, the passing acts like call-by-value. The object reference is passed to the function parameters. They can't be changed within the function, because they can't be changed at all, i.e. they are immutable. It's different, if we pass mutable arguments. They are also passed by object reference, but they can be changed in place in the function. If we pass a list to a function, we have to consider two cases: Elements of a list can be changed in place, i.e. the list will be changed even in the caller's scope. If a new list is assigned to the name, the old list will not be affected, i.e. the list in the caller's scope will remain untouched.

Parameter passing

simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\simple.py (3.4.4)

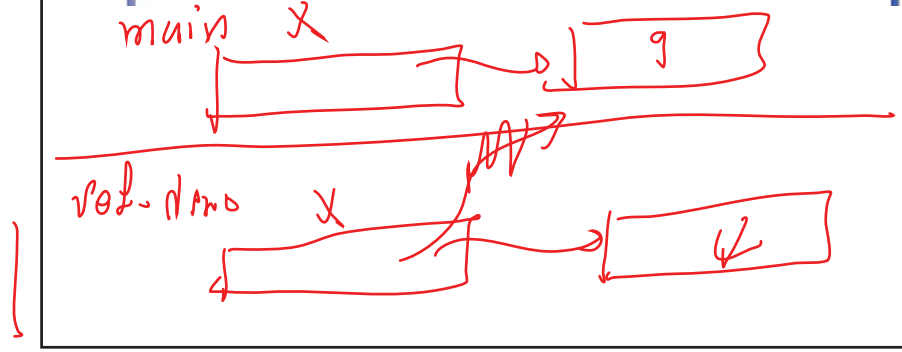
File Edit Format Run Options Window Help

```
def ref_demo(x):
    print ("inside  x=",x)
    x=4
    print ("inside  x=",x)


x = 9
print ("outside x=",x)
ref_demo(x)
print ("outside x=",x)
```

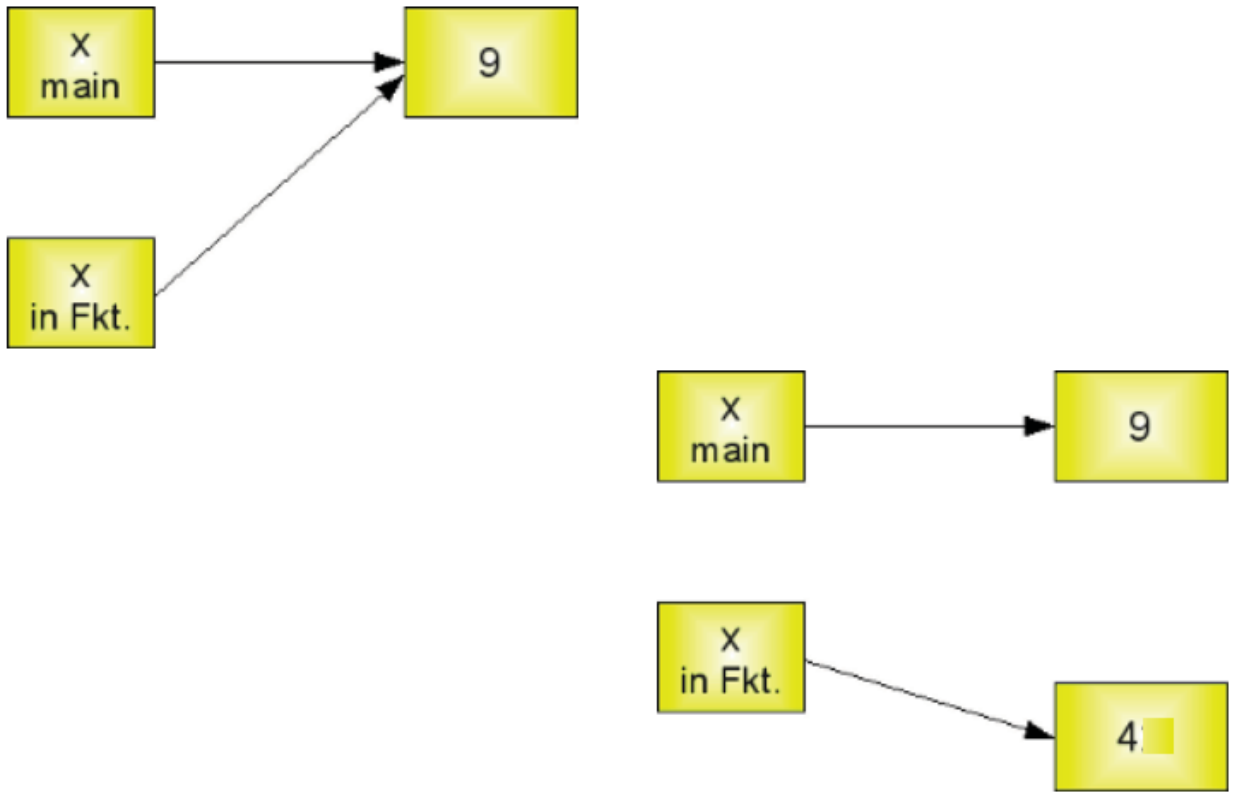
Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:737efcad
tel)] on win32
Type "copyright", "credits" o
>>>
 RESTART: C:\Documents and Se
ple.py
outside x= 9
inside  x= 9
inside  x= 4
outside x= 9
```



Parameter passing



Functions without return

- ***All* functions in Python have a return value**
  - even if no *return* line inside the code.
- **Functions without a *return* return the special value *None*.**
  - *None* is a special constant in the language.
  - *None* is used like *null* in Java.
  - *None* is also logically equivalent to False.

Keyword arguments

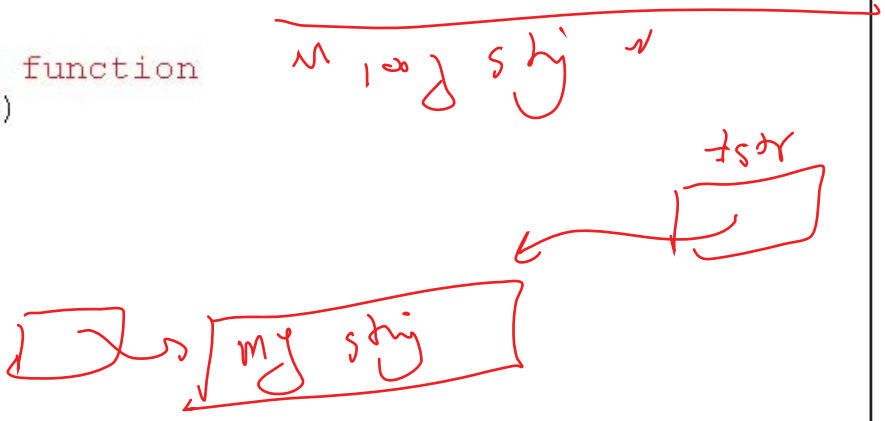
- Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters. You can also make keyword calls to the *printme()* function in the following ways −

Keyword arguments

fuadd.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\9a-1\fuadd.py (3.4.4)

File Edit Format Run Options Window Help

```
# Function definition is here
def printme( tstr ):
   "This prints a passed string into this function"
   print (tstr)
   return;

# Now you can call printme function
printme( tstr = "My string")
```

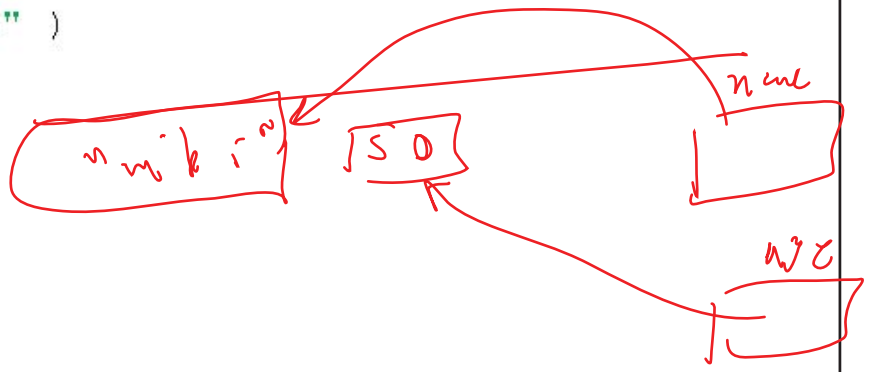


Keyword arguments

fuadd.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\9a-1\fuadd.py (3.4.4)

File Edit Format Run Options Window Help

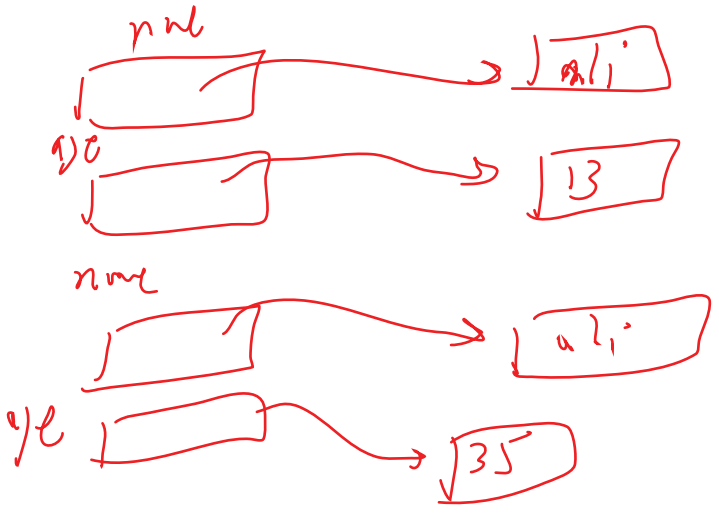
```
# Function definition is here
def printinfo( name, age ):
   "This prints a passed info into this function"
   print ("Name: ", name)
   print ("Age ", age)
   return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
```



Default arguments

- **A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. The following example gives an idea on default arguments, it prints default age if it is not passed −**

```
def  printinfo ( name, age=35)
        print("Name:", name)
        print("age:", age)

printinfo("ali", 13)
printinfo("ali")
```



Default arguments

simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\simple.py (3.4.4)

File Edit Format Run Options Window Help

```
# Function definition is here
def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   print ("Name: ", name)
   print ("Age ", age)
   return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```