Function 2

Parameterised function

by

Lilian Blot

Learning Objectives

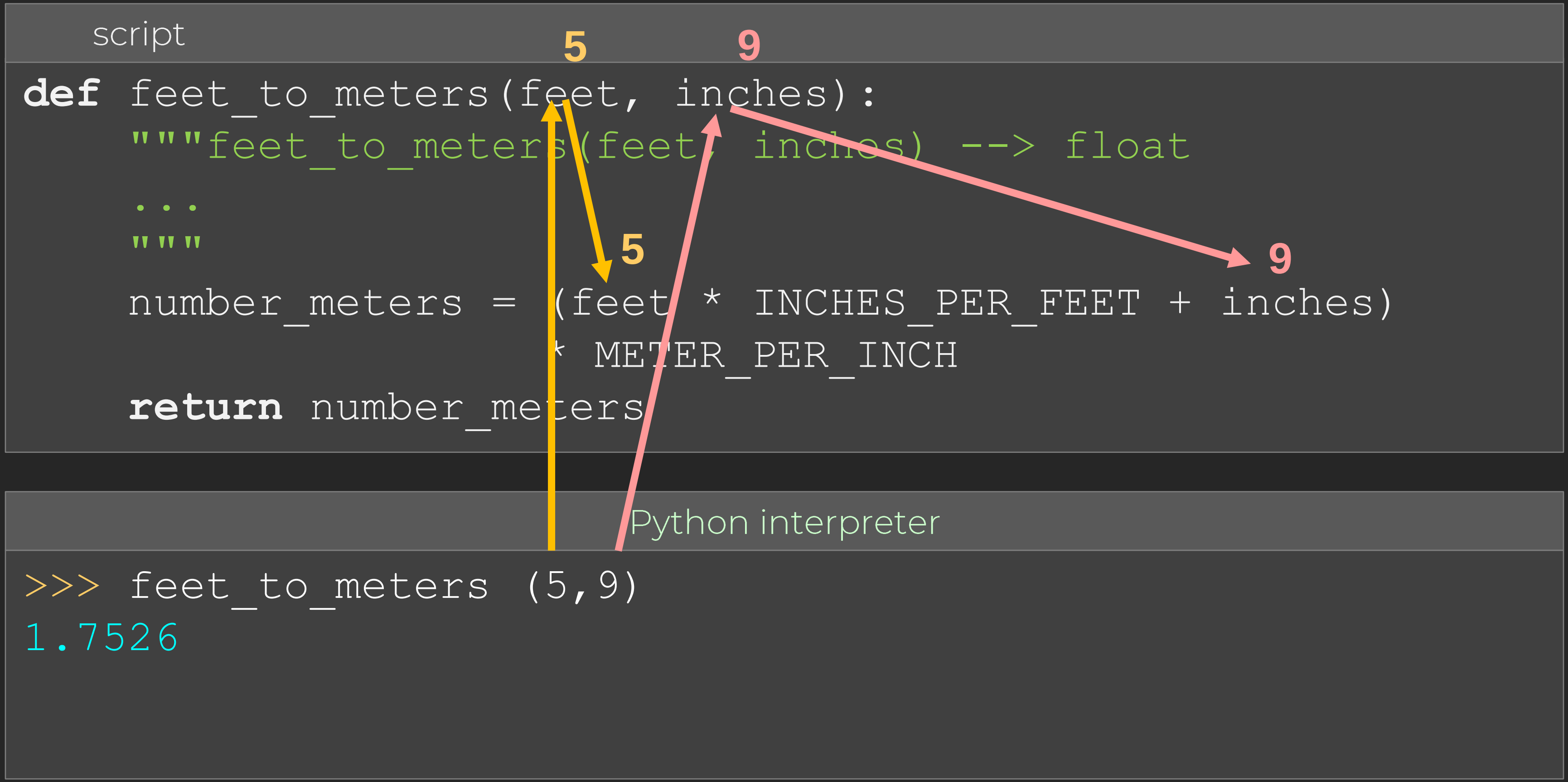
Call function using positional and keyword arguments.

Understand and explain how mutable and immutable objects are passed in parameters.

Positional Arguments & Keyword Arguments

Calling a function with parameters

Positional Arguments matching: matched left to right



script

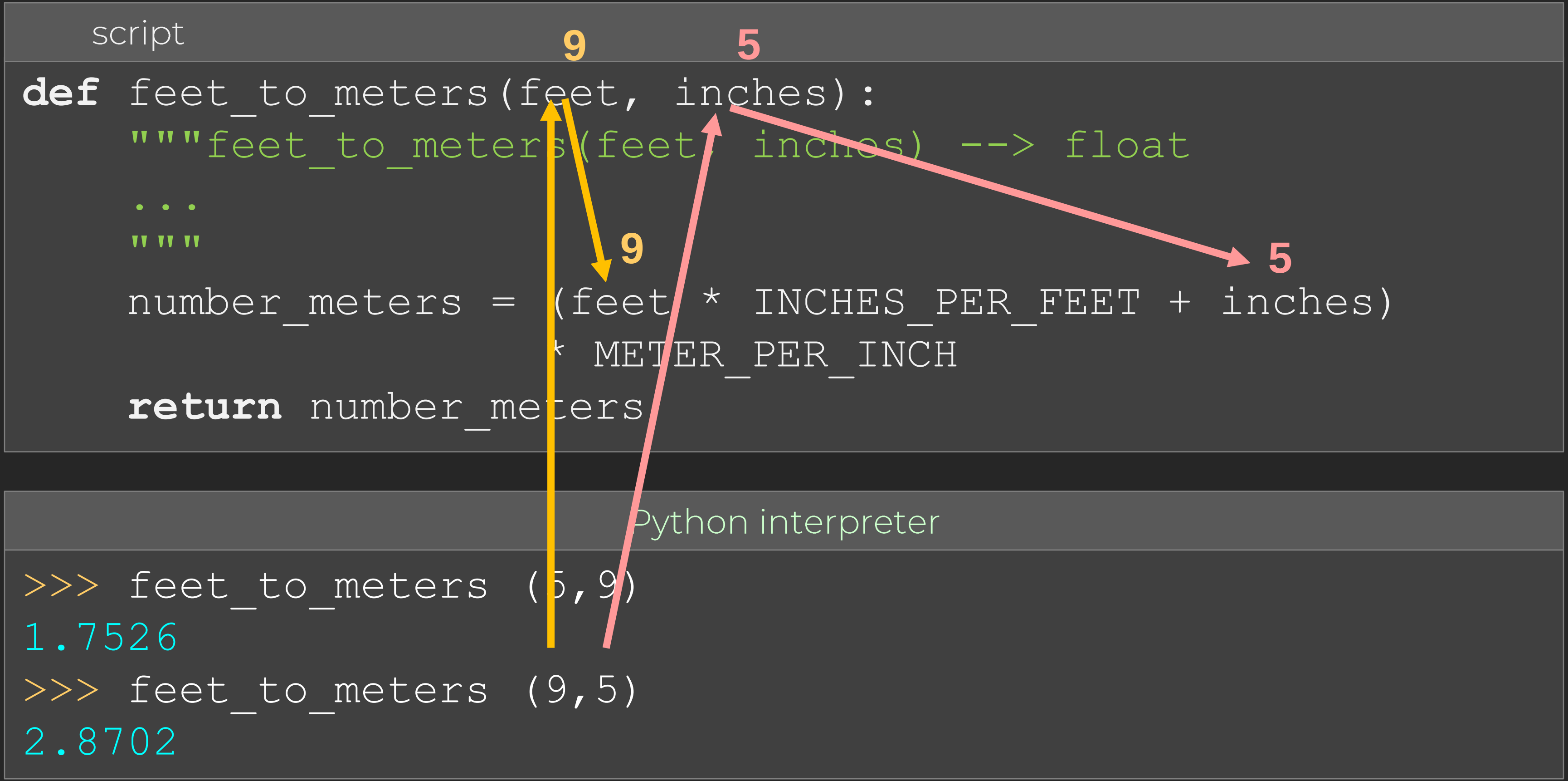
```
def feet_to_meters(feet, inches):
    """feet_to_meters(feet, inches) --> float
    ...
    """
    number_meters = (feet * INCHES_PER_FEET + inches)
                    * METER_PER_INCH
    return number_meters
```

Python interpreter

```
>>> feet_to_meters (5,9)
1.7526
```

Calling a function with parameters

Positional Arguments matching: matched left to right



script

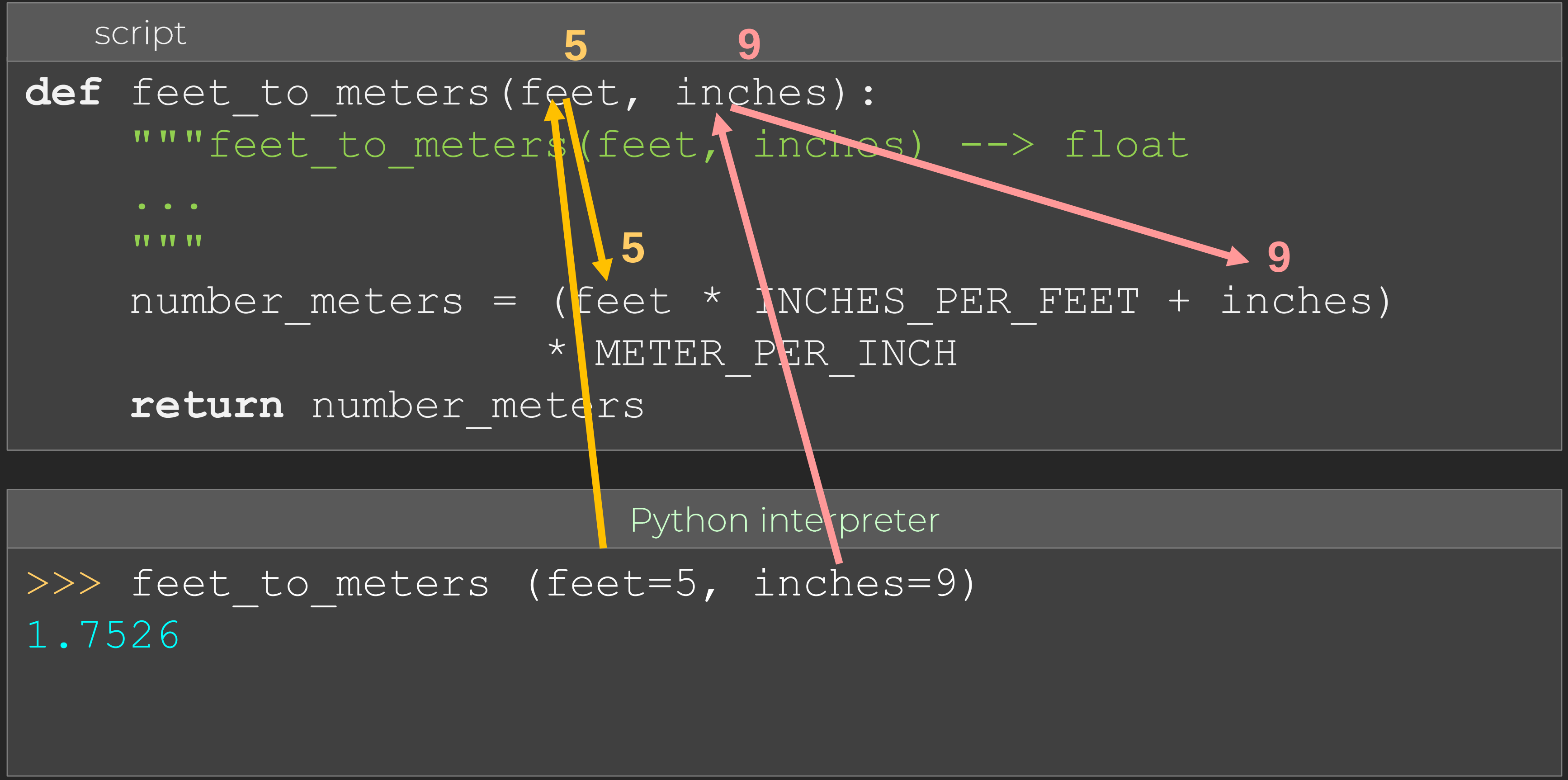
```
def feet_to_meters(feet, inches):
    """feet_to_meters(feet, inches) --> float

    ...
    """
    number_meters = (feet * INCHES_PER_FEET + inches)
                    * METER_PER_INCH
    return number_meters
```

Python interpreter

```
>>> feet_to_meters (5,9)
1.7526
>>> feet_to_meters (9,5)
2.8702
```

Calling a function with parameters

Keywords Arguments matching: matched by argument's name



script

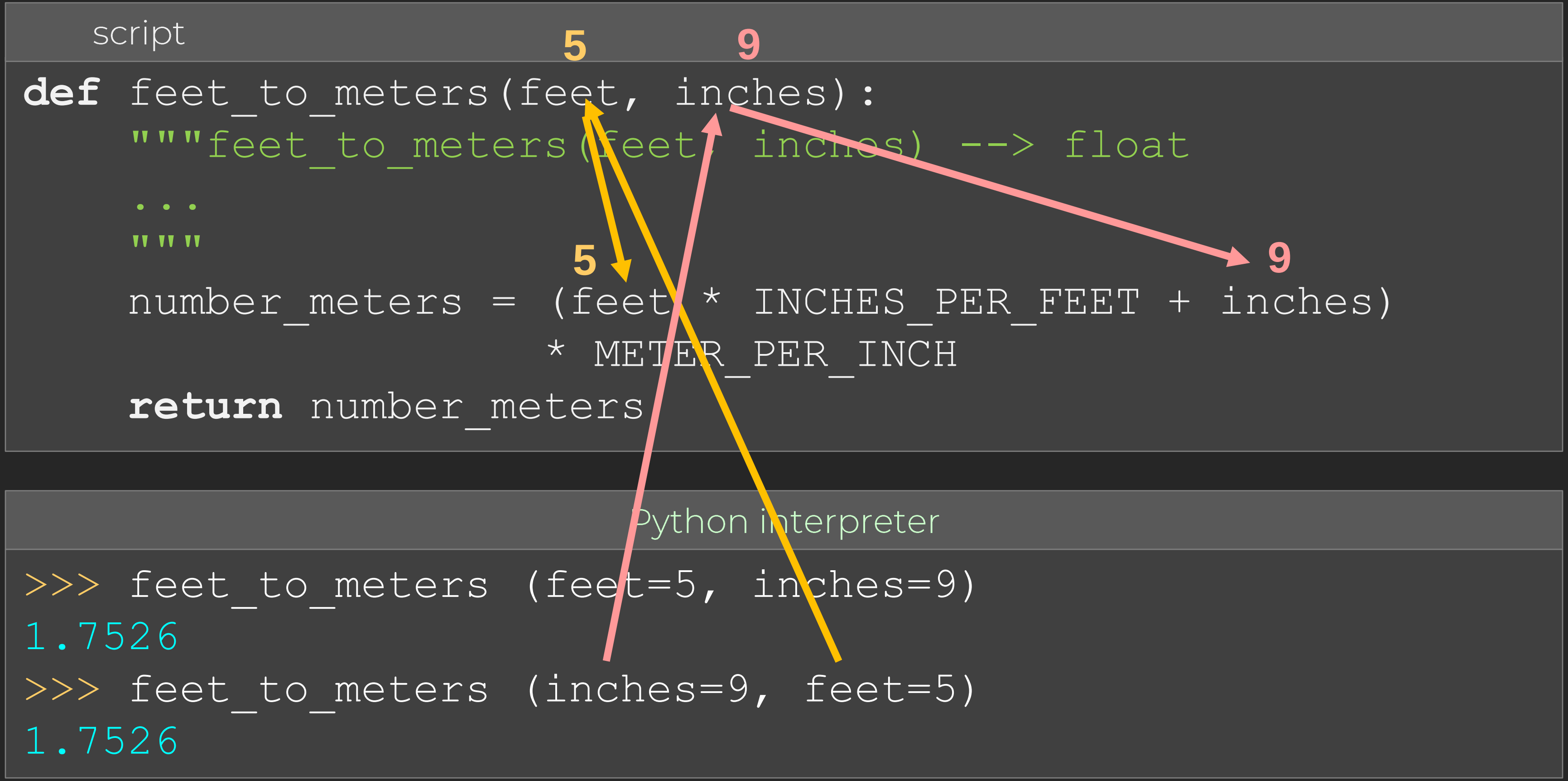
```
def feet_to_meters(feet, inches):
    """feet_to_meters(feet, inches) --> float
    ...
    """
    number_meters = (feet * INCHES_PER_FEET + inches)
                    * METER_PER_INCH
    return number_meters
```

Python interpreter

```
>>> feet_to_meters (feet=5, inches=9)
1.7526
```

Calling a function with parameters

Keywords Arguments matching: matched by argument’s name



script

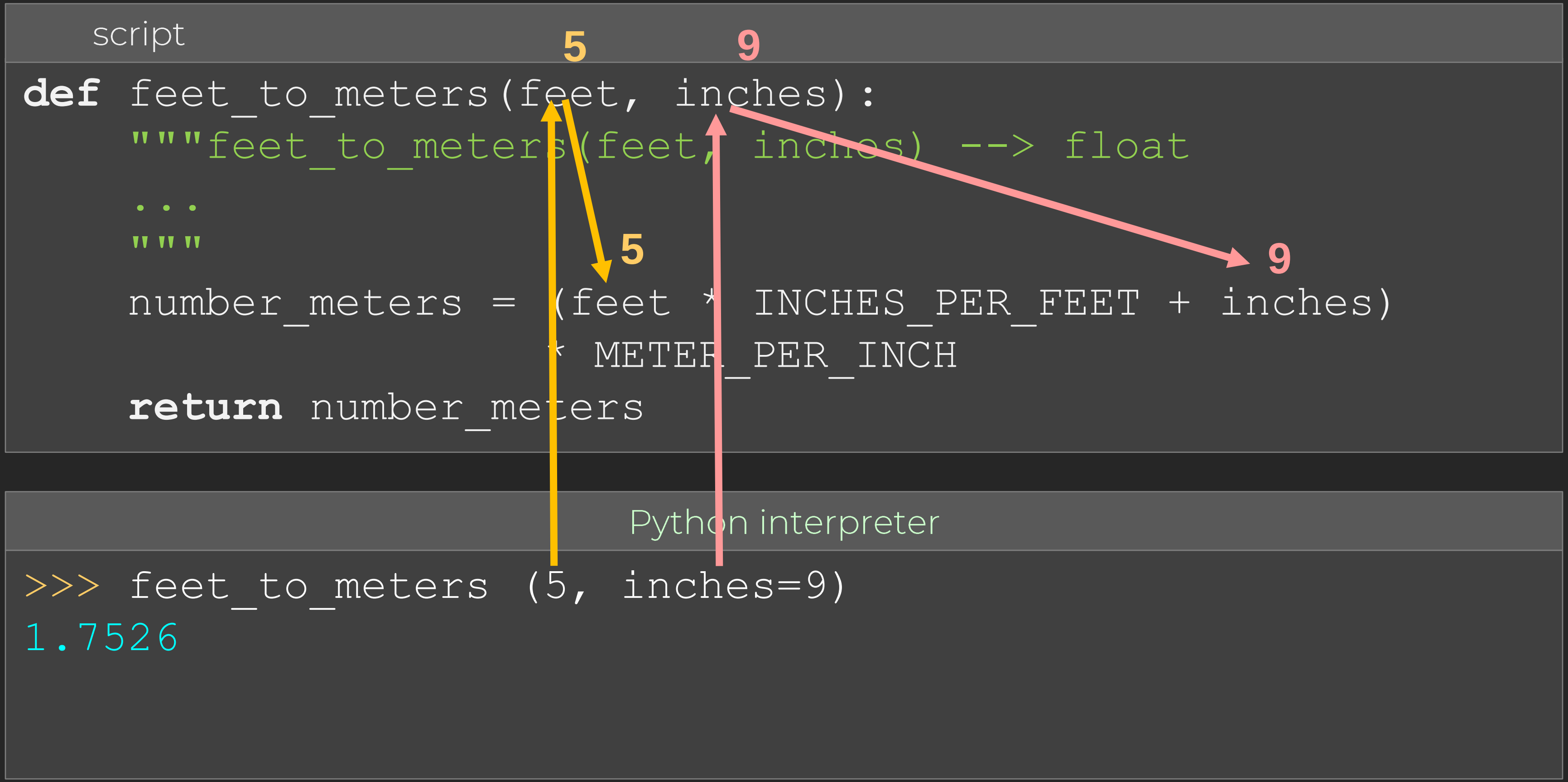
```
def feet_to_meters(feet, inches):
    """feet_to_meters(feet, inches) --> float
    ...
    """
    number_meters = (feet * INCHES_PER_FEET + inches)
                    * METER_PER_INCH
    return number_meters
```

Python interpreter

```
>>> feet_to_meters (feet=5, inches=9)
1.7526
>>> feet_to_meters (inches=9, feet=5)
1.7526
```

Calling a function with parameters

Mixed Positional & Keywords Arguments matching



```
def feet_to_meters(feet, inches):
    """feet_to_meters(feet, inches) --> float
    ...
    """
    number_meters = (feet * INCHES_PER_FEET + inches)
                    * METER_PER_INCH
    return number_meters
```

```
>>> feet_to_meters (5, inches=9)
1.7526
```

Calling a function with parameters

Mixed Positional & Keywords Arguments matching

script

```python
def feet_to_meters(feet, inches):
    """feet_to_meters(feet, inches) --> float
    ...
    """
    number_meters = (feet * INCHES_PER_FEET + inches)
                    * METER_PER_INCH
    return number_meters
```

Python interpreter

```
>>> feet_to_meters (5, inches=9)
1.7526
>>> feet_to_meters (inches=9,5)
SyntaxError: positional argument follows keyword argument
```

Calling a function with parameters

Mixed Positional & Keywords Arguments matching

script

```
def feet_to_meters(feet, inches):
    """feet_to_meters(feet, inches)
    ...
    """
    number_me                              _FEET + inches)
                                 _INCH
```

Be careful, positional arguments MUST precede keyword arguments

Python interpreter

```
>>> feet_to_meters (5, inches=9)
1.7526
>>> feet_to_meters (inches=9,5)
SyntaxError: positional argument follows keyword argument
```

Passing immutable objects in Parameters

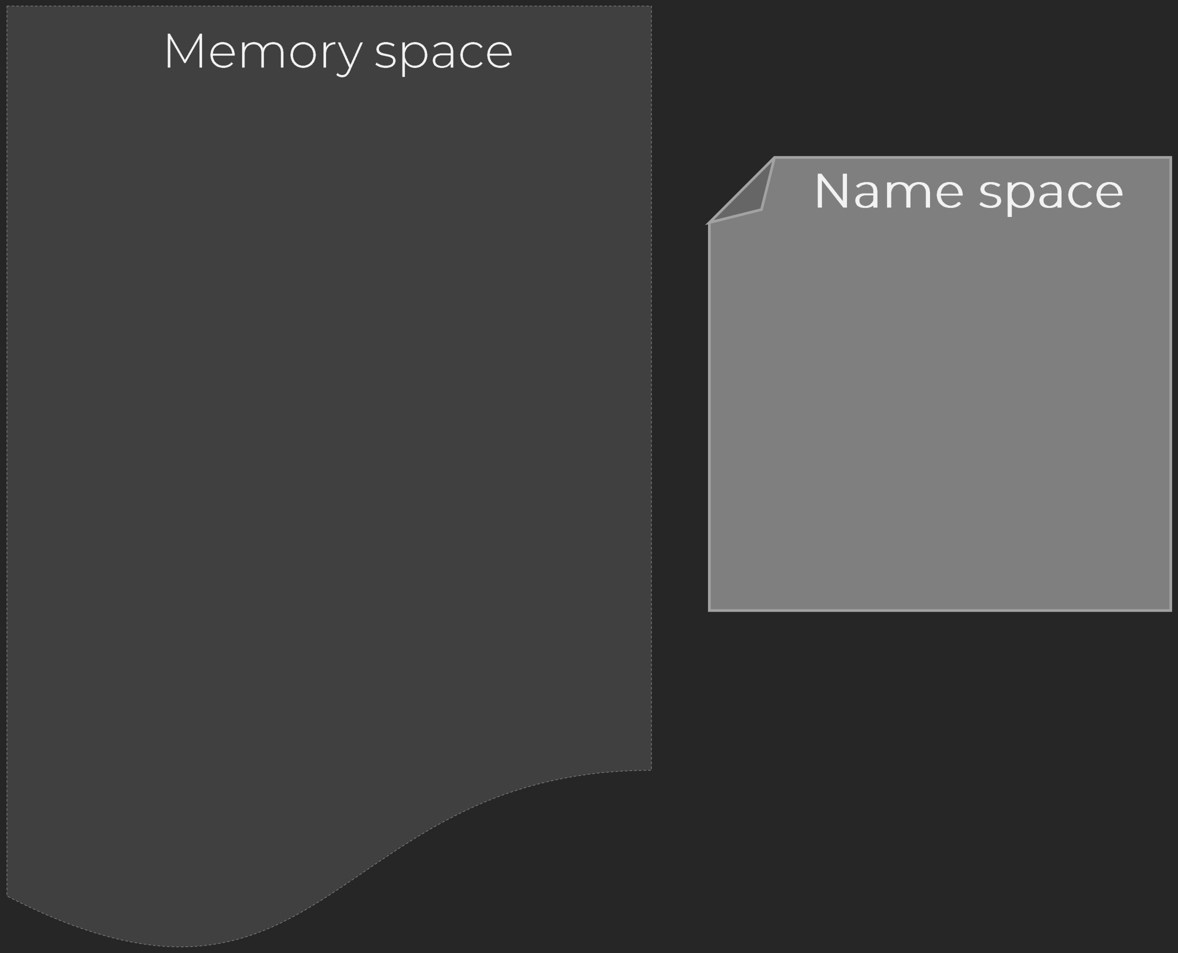
When Python comes to a function call, it initiate a four-step process:

1. the calling program suspends execution at the point of call
2. the formal parameter of the function get assigned the value supplied by the actual parameters in the call
3. the body of the function is executed
4. control returns to the point just after where the function was called

Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)

my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

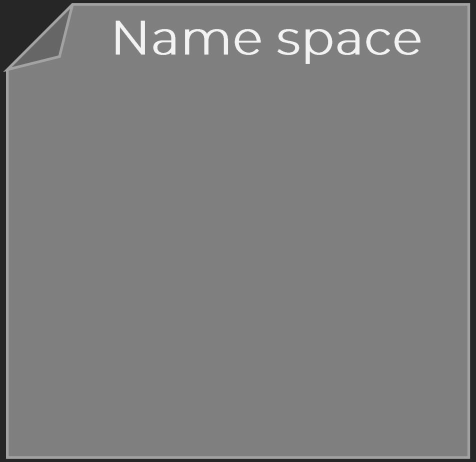
Python shell



Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)

my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

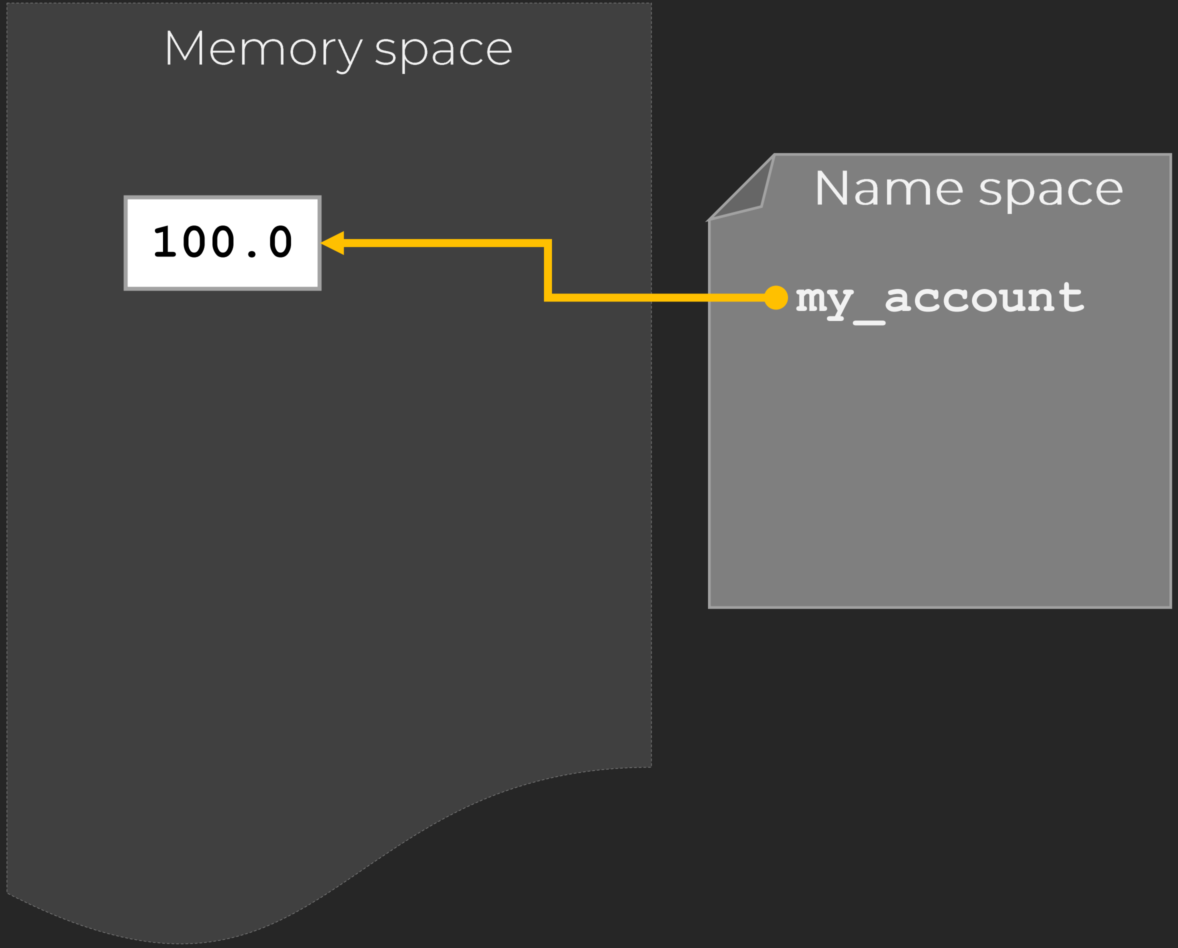
Python shell



Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)

my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

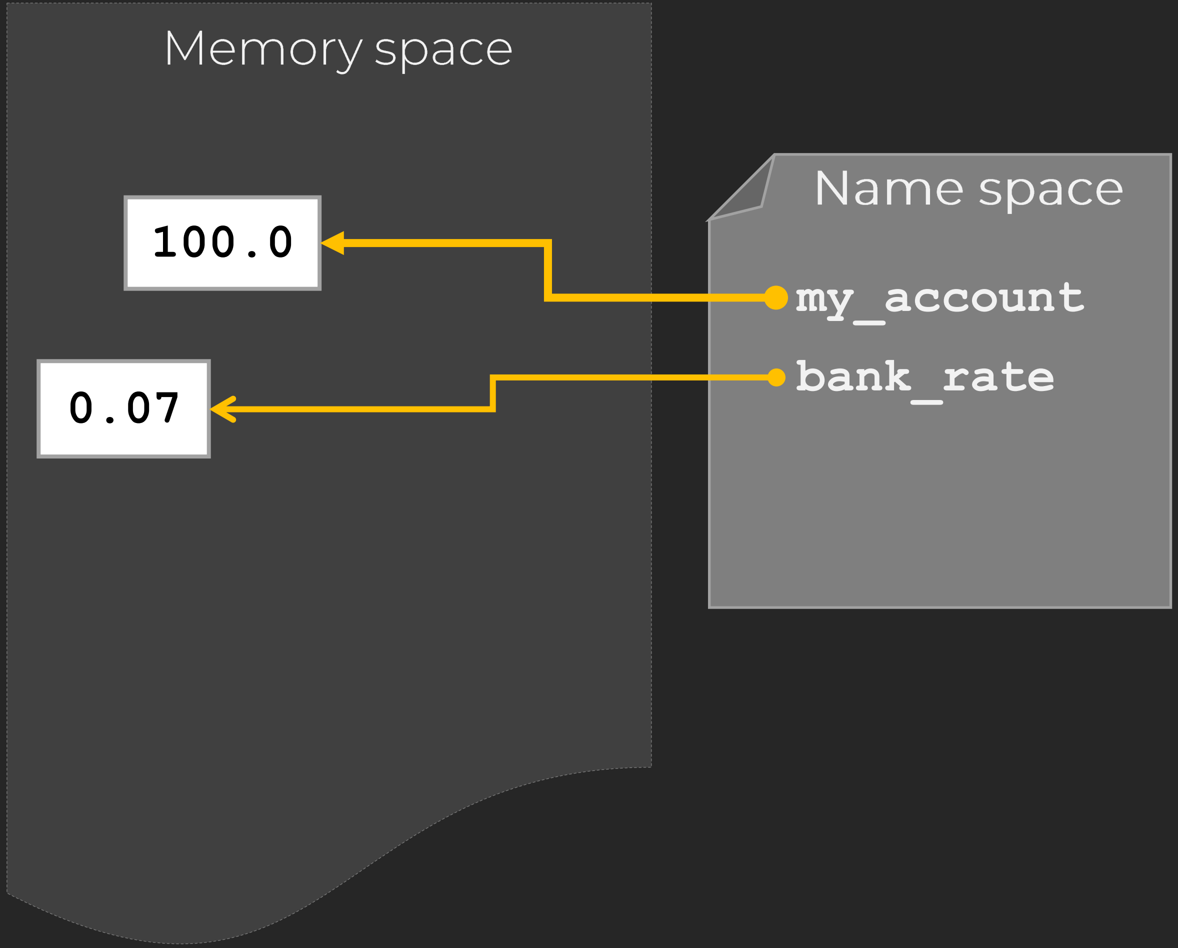
Python shell



Code

```python
def addInterestOne(account, rate):
    account =  account * (1+rate)

my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

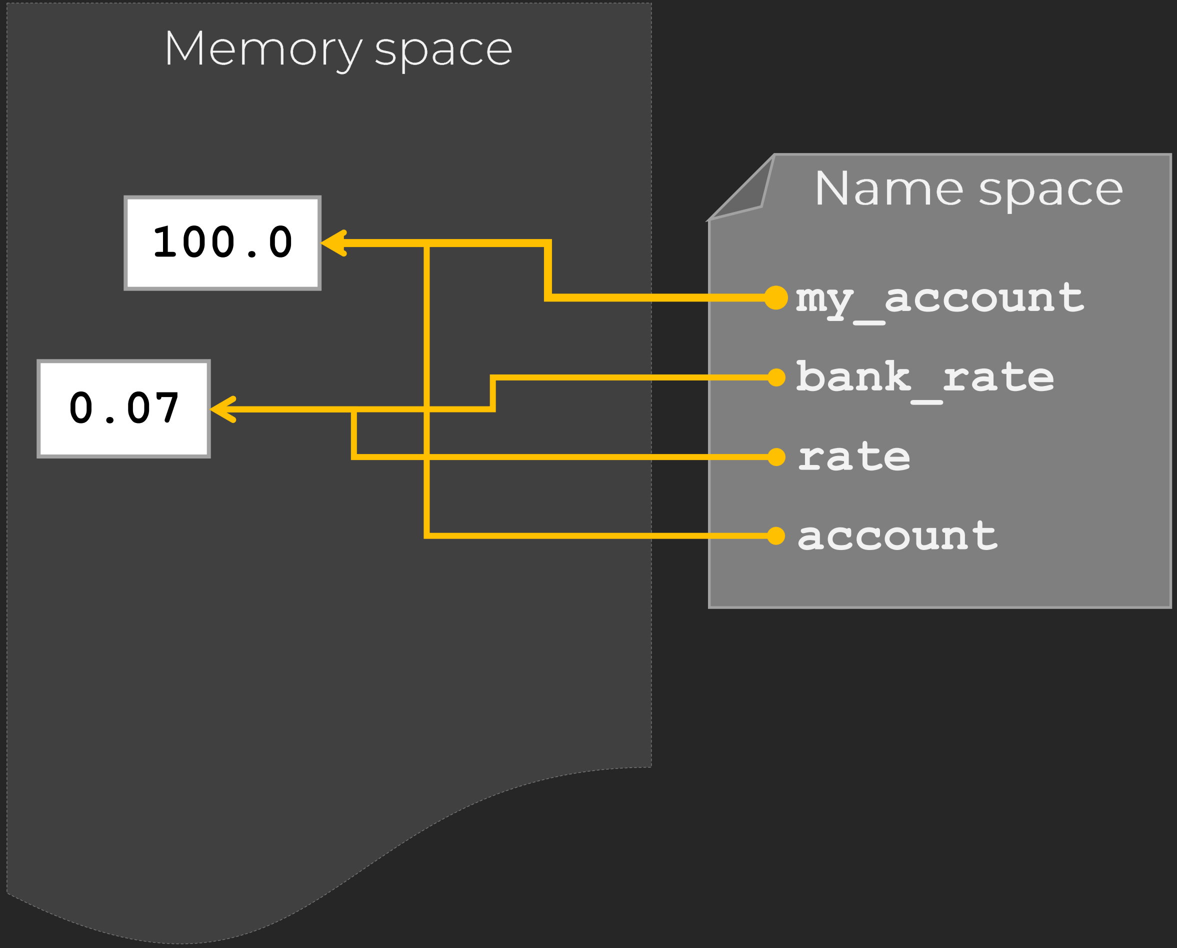
Python shell



Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)

my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

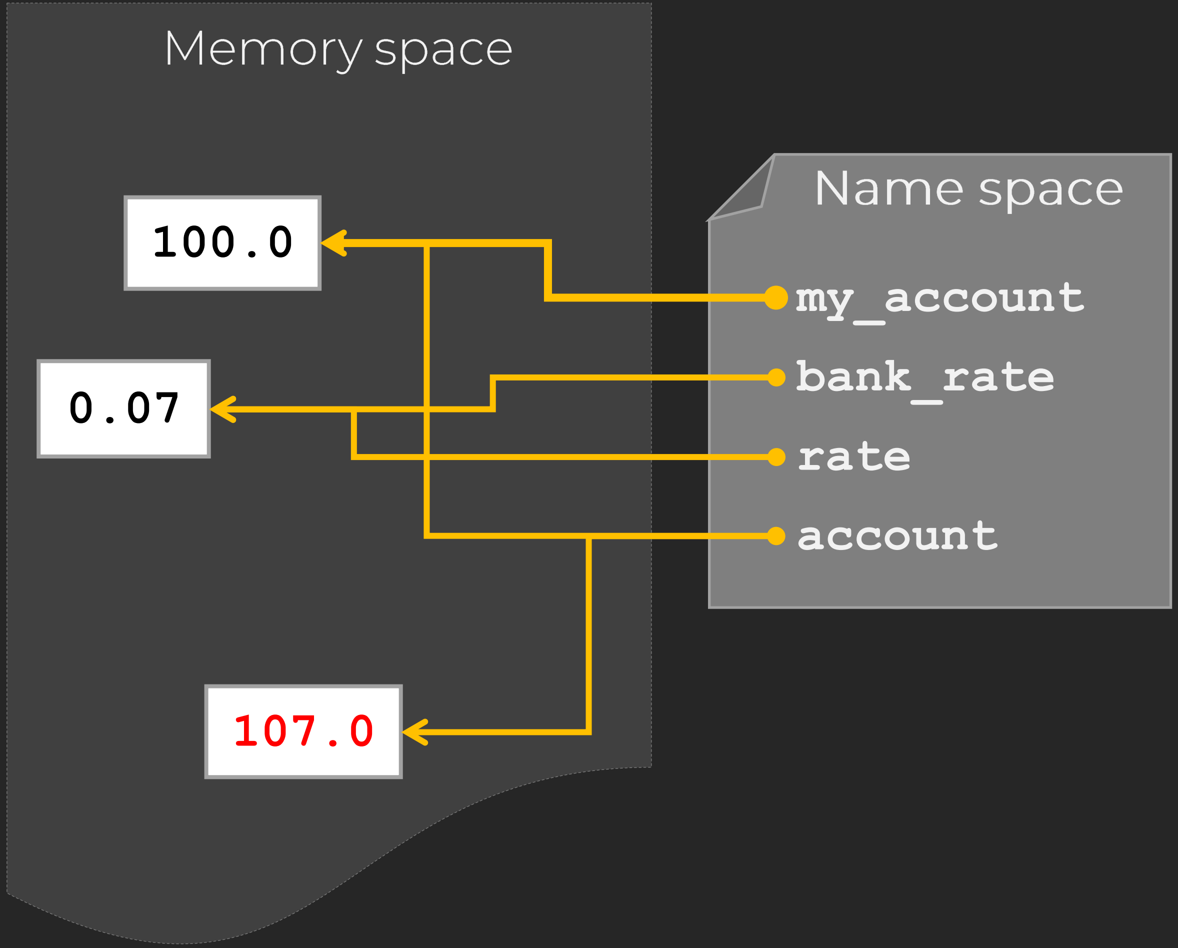
Python shell



Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)

my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

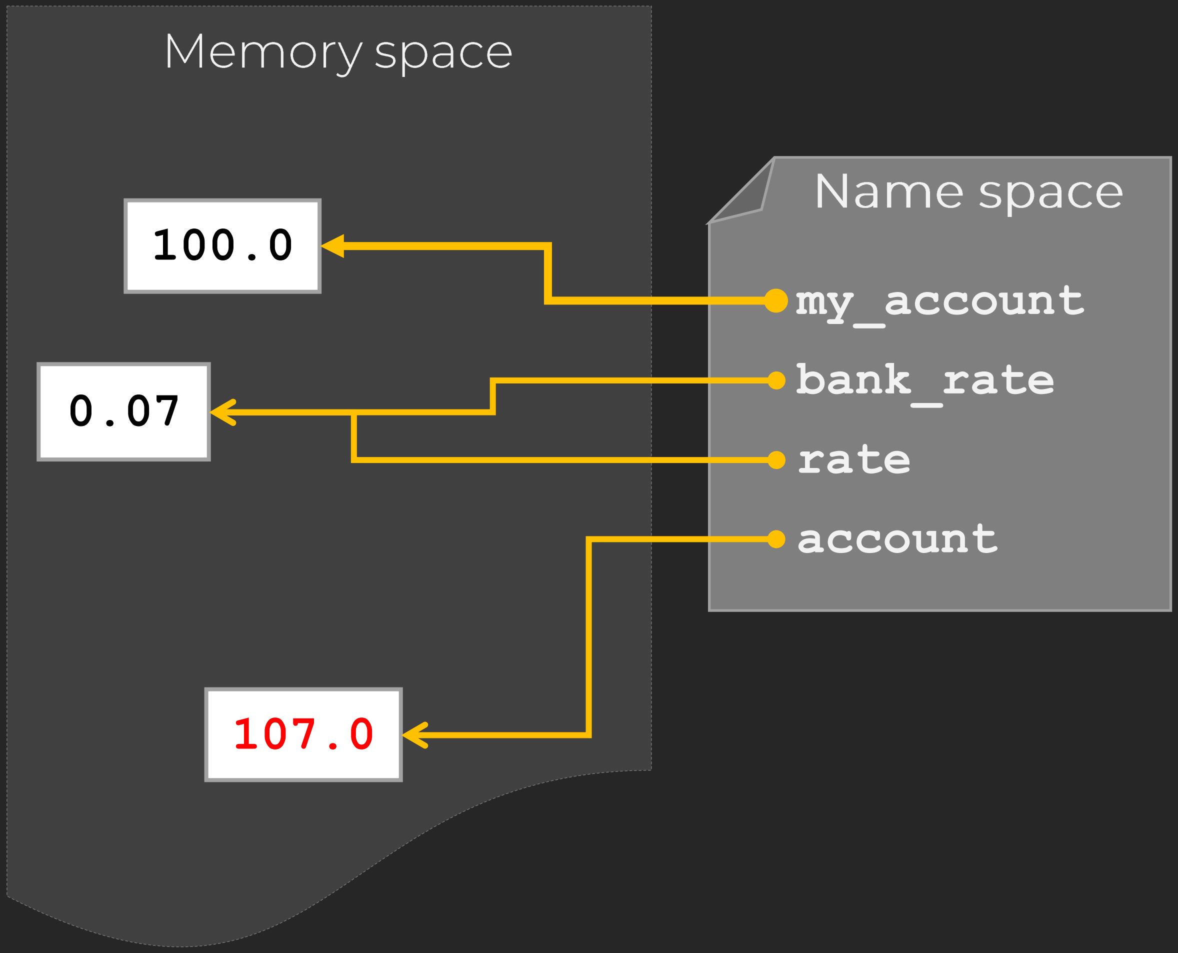
Python shell



Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)

my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

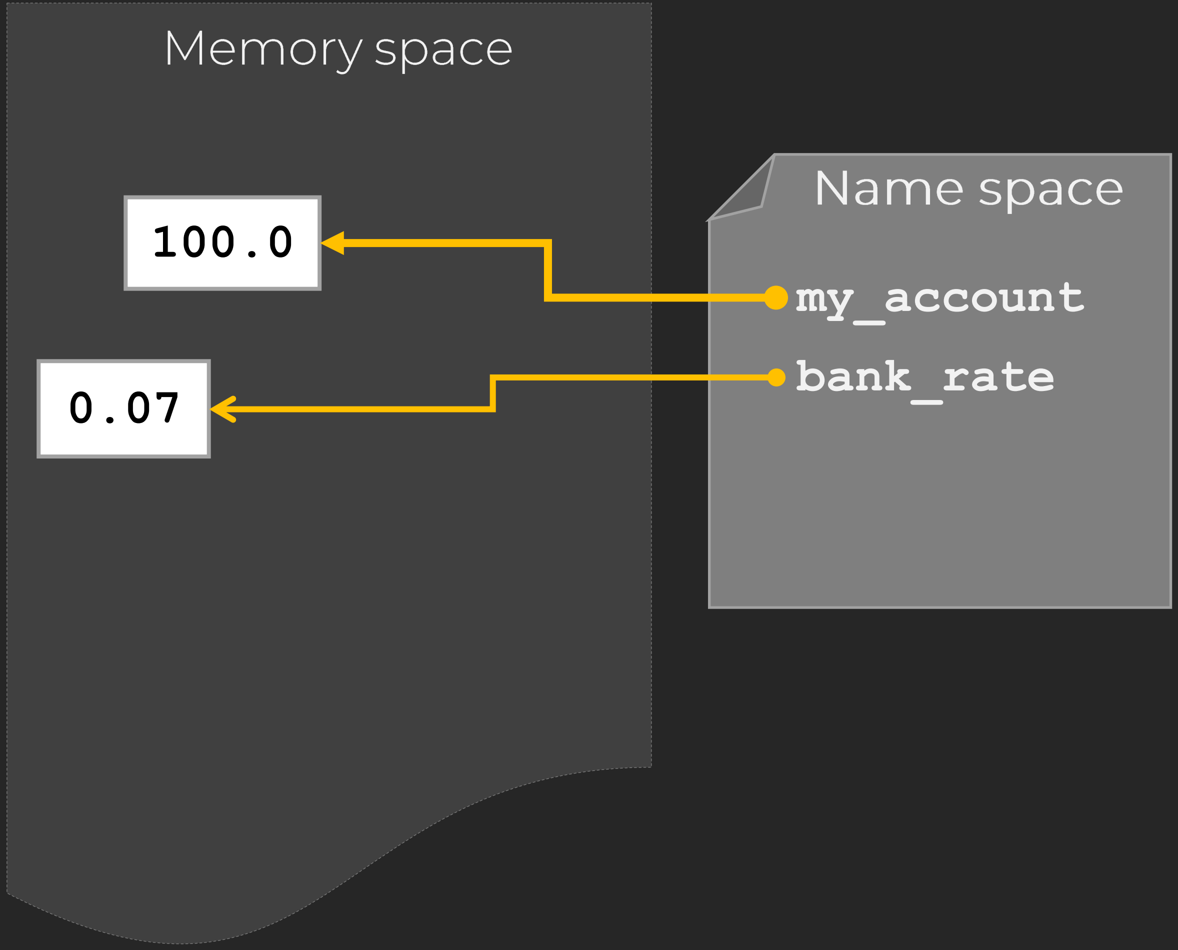
Python shell



Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)

my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

Python shell

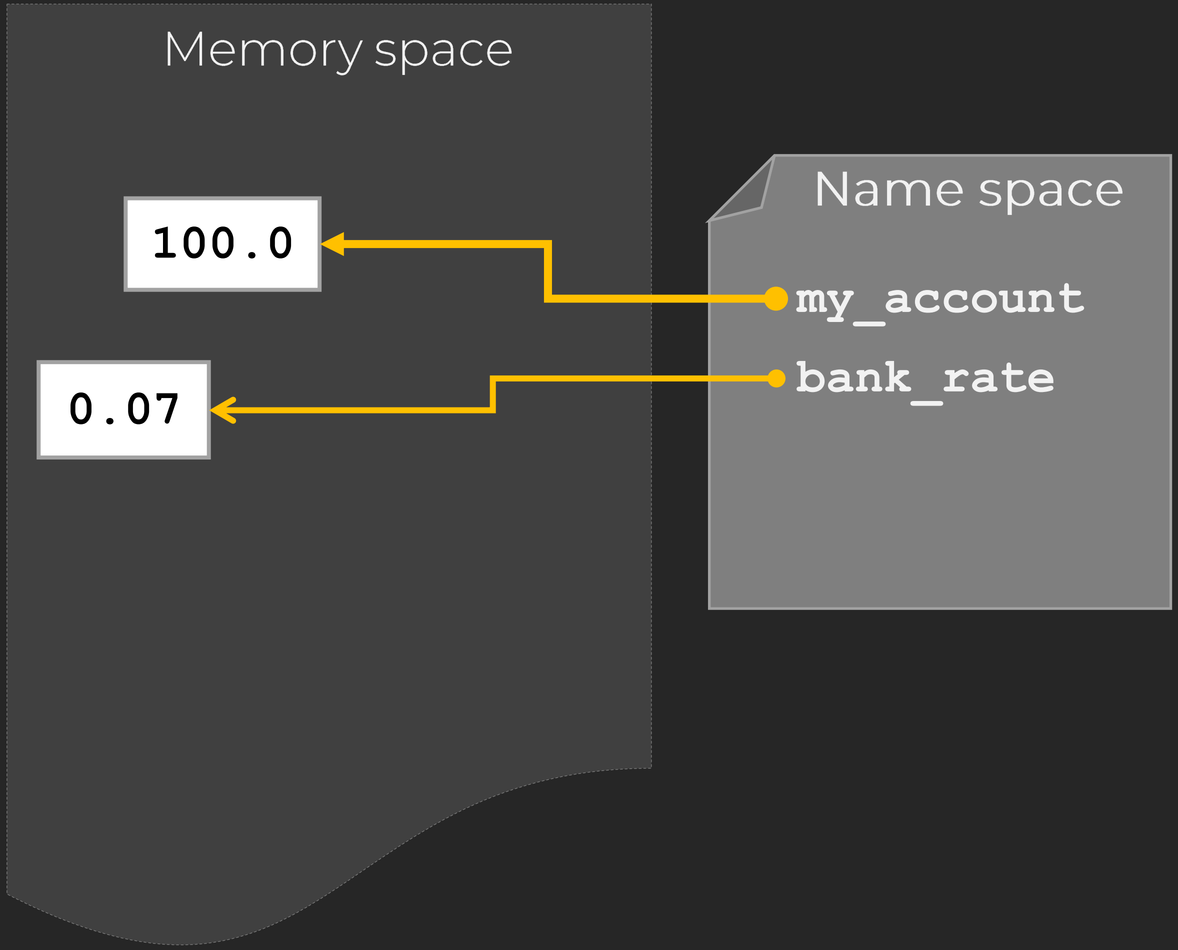


Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)

my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

Python shell

```
new accounts balance:100
```



Return a value instead

Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)



my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

Return a value instead

Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)
    return account

my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

Return a value instead

Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)
    return account

my_account = 100.0
bank_rate = 0.07
my_account = addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

Return a value instead

Code

```
def addInterestOne(account, rate):
    account =  account * (1+rate)
    return account

my_account = 100.0
bank_rate = 0.07
my_account = addInterestOne(my_account, bank_rate)
print("new accounts balance:",\
      my_account)
```

Python shell

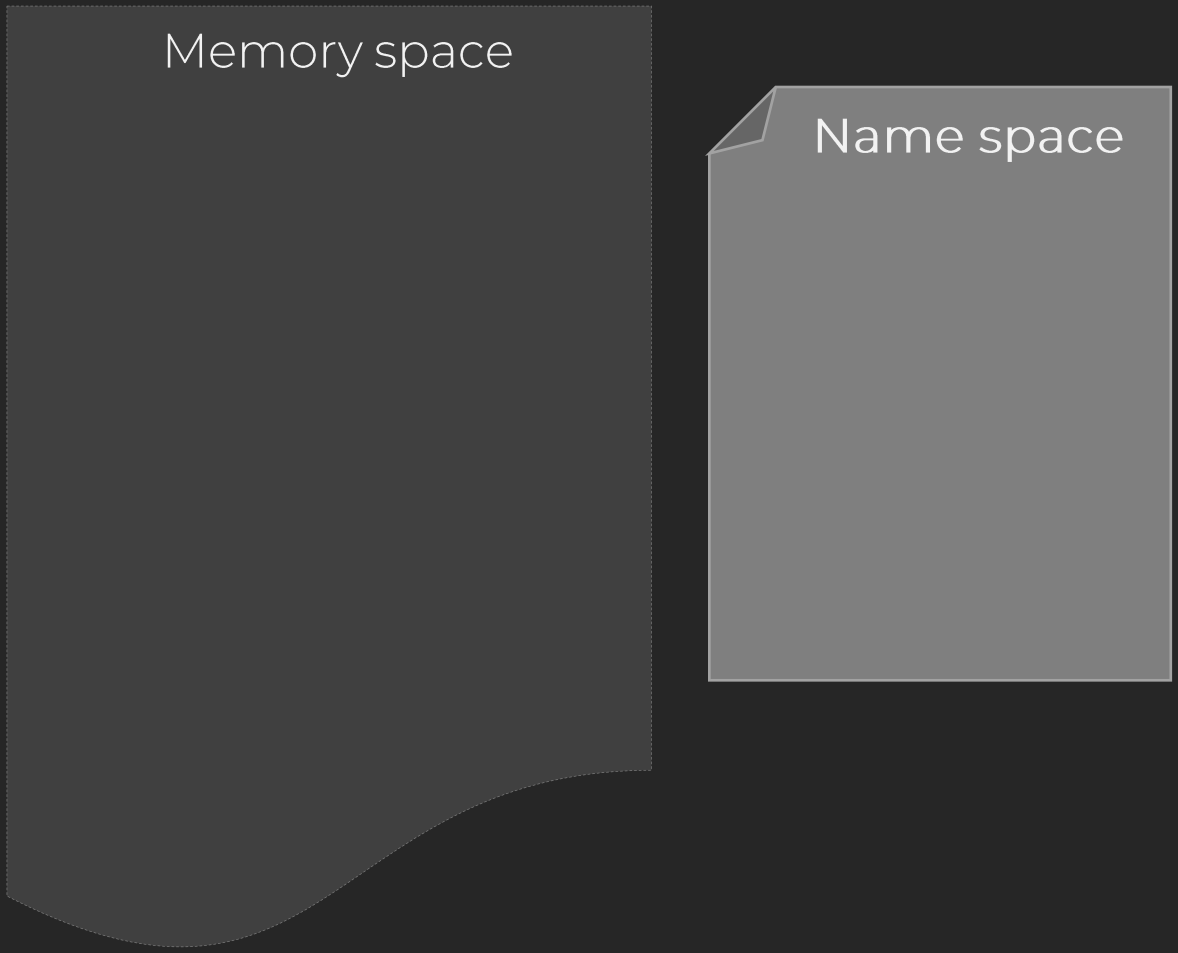
```
new accounts balance:107
```

Passing mutable objects in Parameters

Code

```
def addInterestAll(accounts, rate):
    for i in range(len(accounts)):
        accounts[i] *= (1+rate)

lst account = [10.0, 20.0, 100.0]
bank_rate = 0.07
addInterestAll(lst_accounts, bank_rate)
print("new accounts balance:",\
      lst_accounts)
```

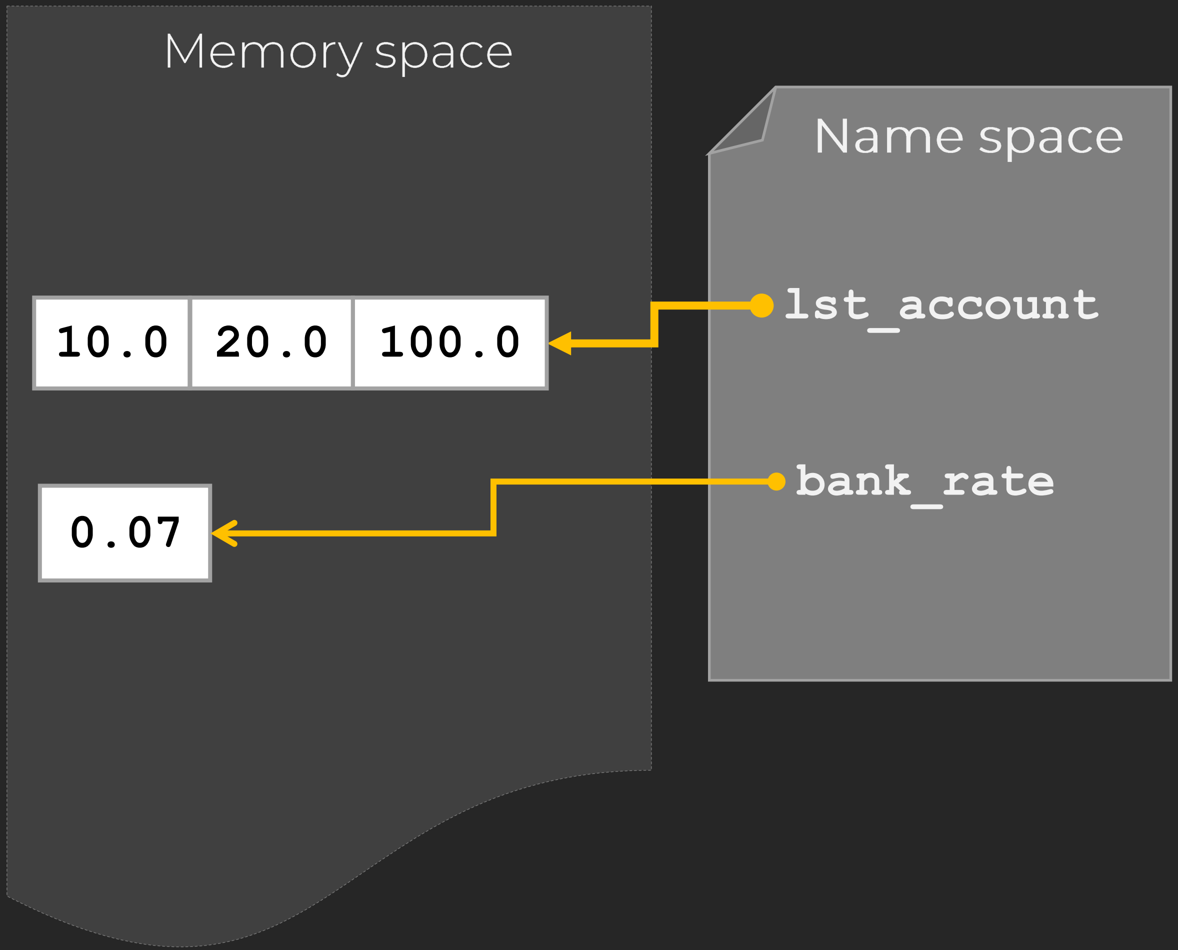
Python shell



Code

```python
def addInterestAll(accounts, rate):
    for i in range(len(accounts)):
        accounts[i] *= (1+rate)

lst_account = [10.0, 20.0, 100.0]
bank_rate = 0.07
addInterestAll(lst_accounts, bank_rate)
print("new accounts balance:",\
      lst_accounts)
```

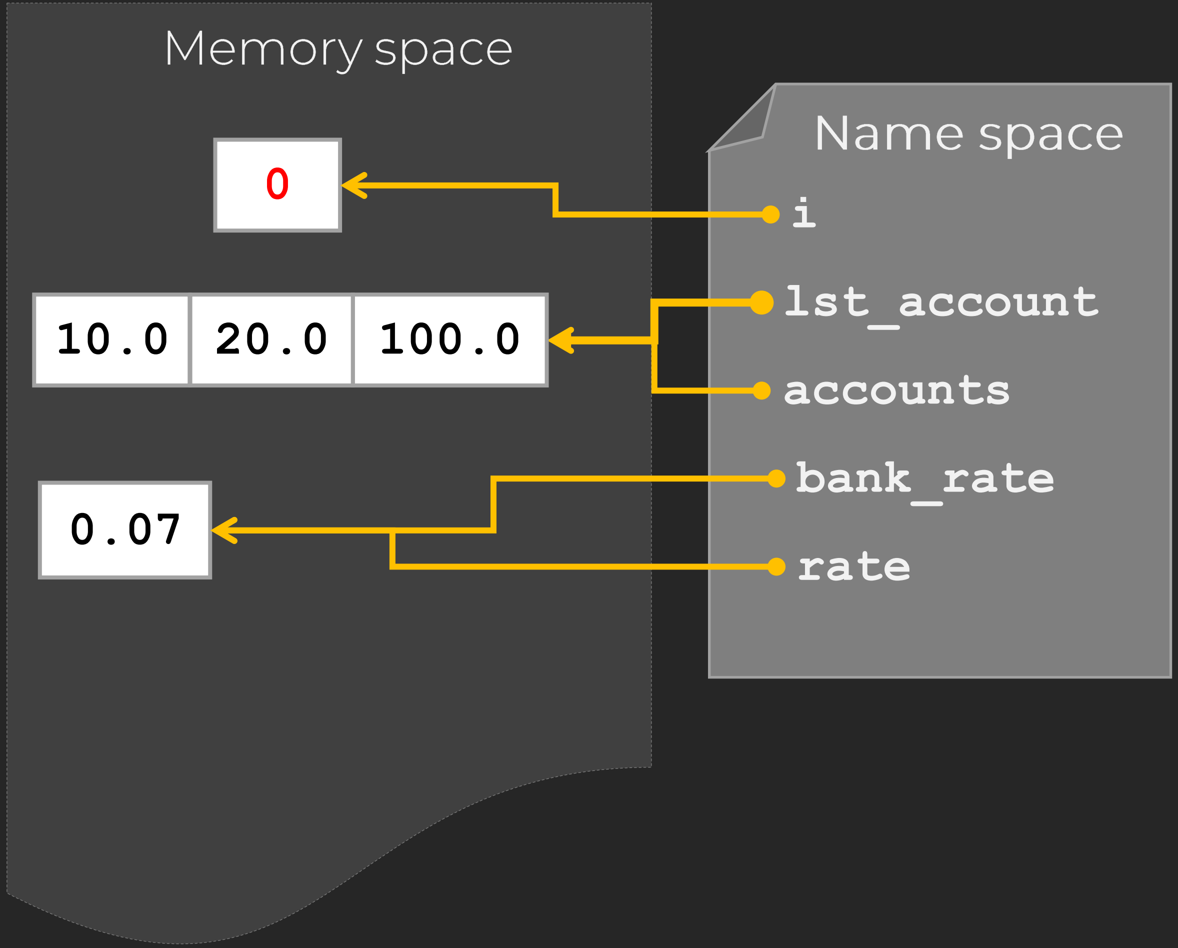
Python shell



Code

```
def addInterestAll(accounts, rate):
    for i in range(len(accounts)):
        accounts[i] *= (1+rate)

lst_account = [10.0, 20.0, 100.0]
bank_rate = 0.07
addInterestAll(lst_accounts, bank_rate)
print("new accounts balance:",\
      lst_accounts)
```

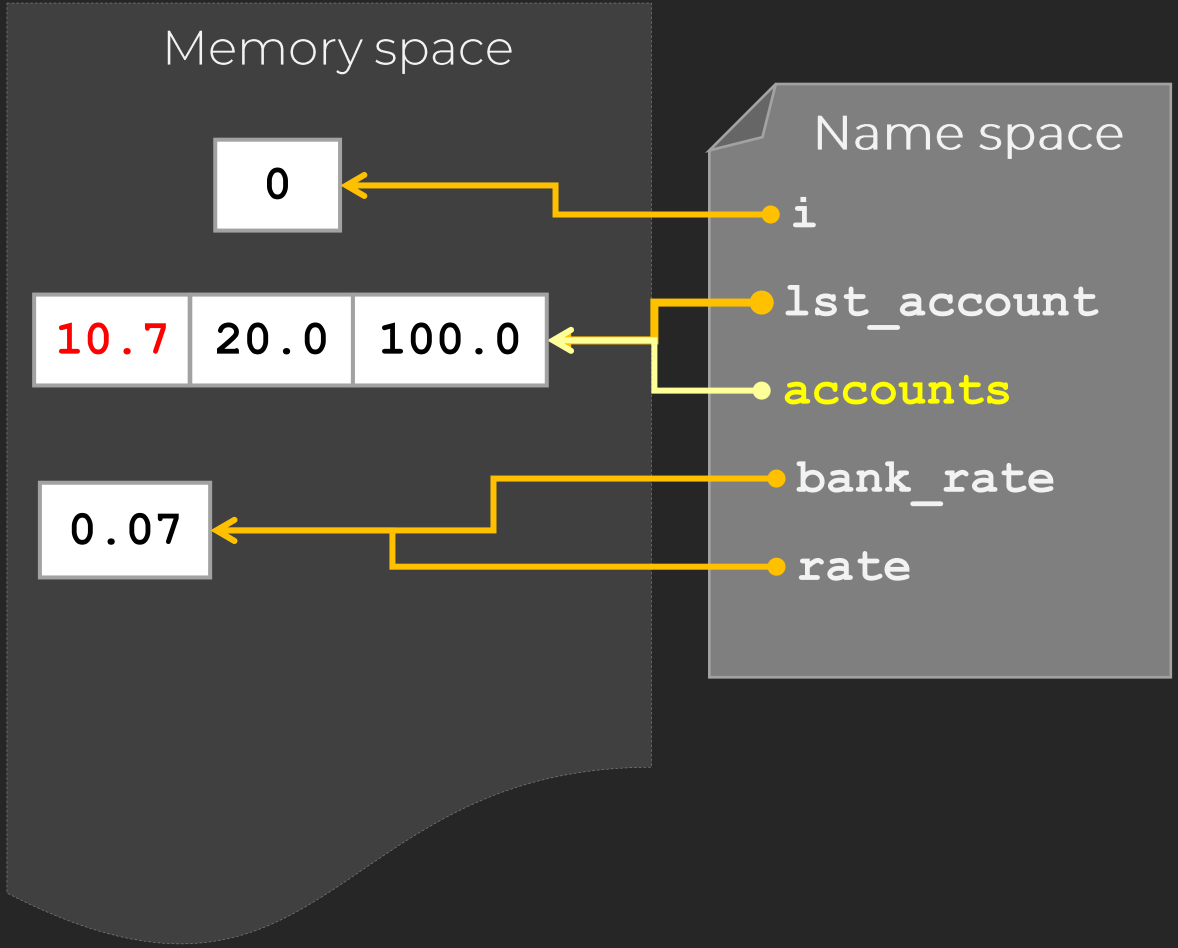
Python shell



Code

```
def addInterestAll(accounts, rate):
    for i in range(len(accounts)):
        accounts[i] *= (1+rate)

lst_account = [10.0, 20.0, 100.0]
bank_rate = 0.07
addInterestAll(lst_accounts, bank_rate)
print("new accounts balance:",\
      lst_accounts)
```

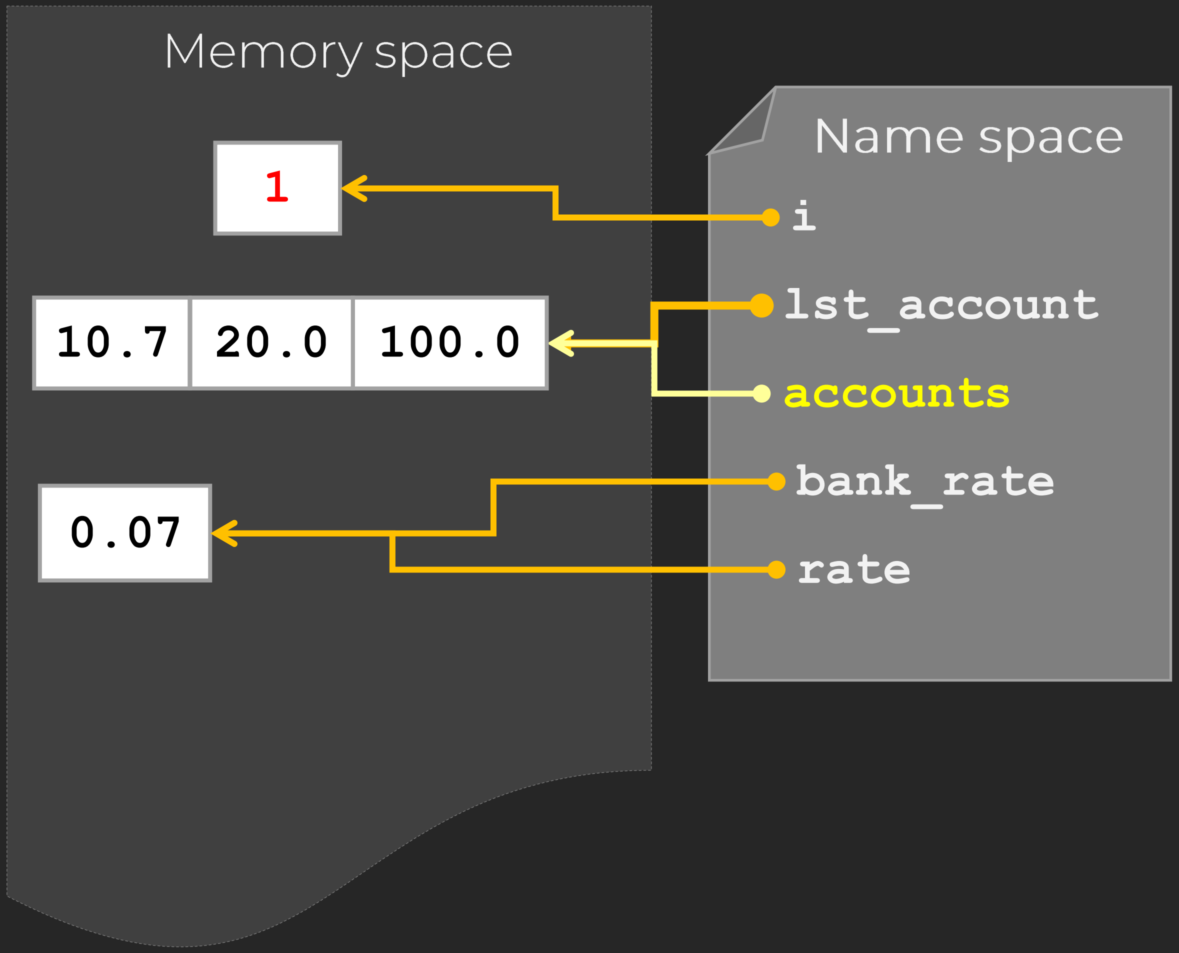
Python shell



Code

```
def addInterestAll(accounts, rate):
    for i in range(len(accounts)):
        accounts[i] *= (1+rate)

lst_account = [10.0, 20.0, 100.0]
bank_rate = 0.07
addInterestAll(lst_accounts, bank_rate)
print("new accounts balance:",\
     lst_accounts)
```

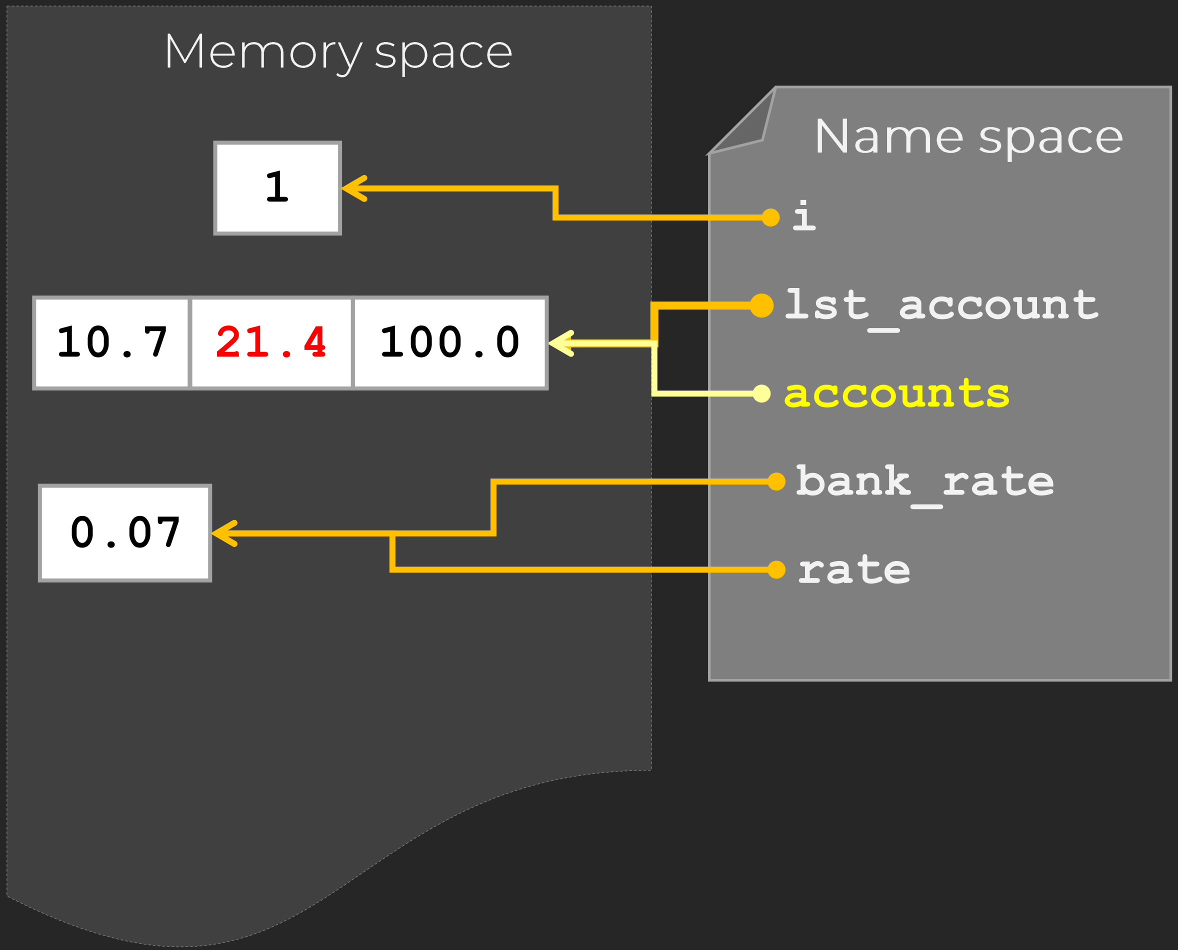
Python shell



Code

```
def addInterestAll(accounts, rate):
    for i in range(len(accounts)):
        accounts[i] *= (1+rate)

lst_account = [10.0, 20.0, 100.0]
bank_rate = 0.07
addInterestAll(lst_accounts, bank_rate)
print("new accounts balance:",\
     lst_accounts)
```

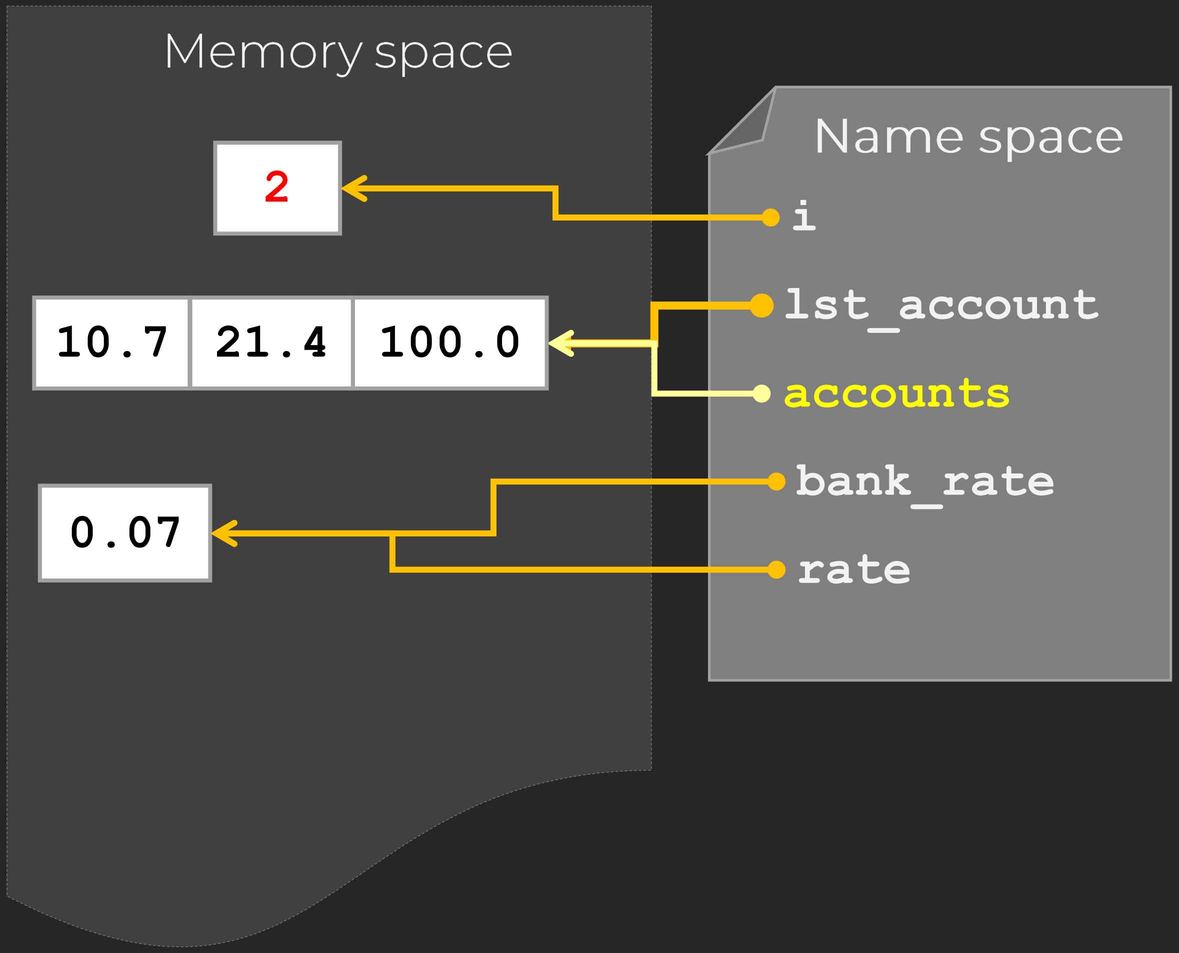
Python shell



Code

```
def addInterestAll(accounts, rate):
    for i in range(len(accounts)):
        accounts[i] *= (1+rate)

lst_account = [10.0, 20.0, 100.0]
bank_rate = 0.07
addInterestAll(lst_accounts, bank_rate)
print("new accounts balance:",\
      lst_accounts)
```

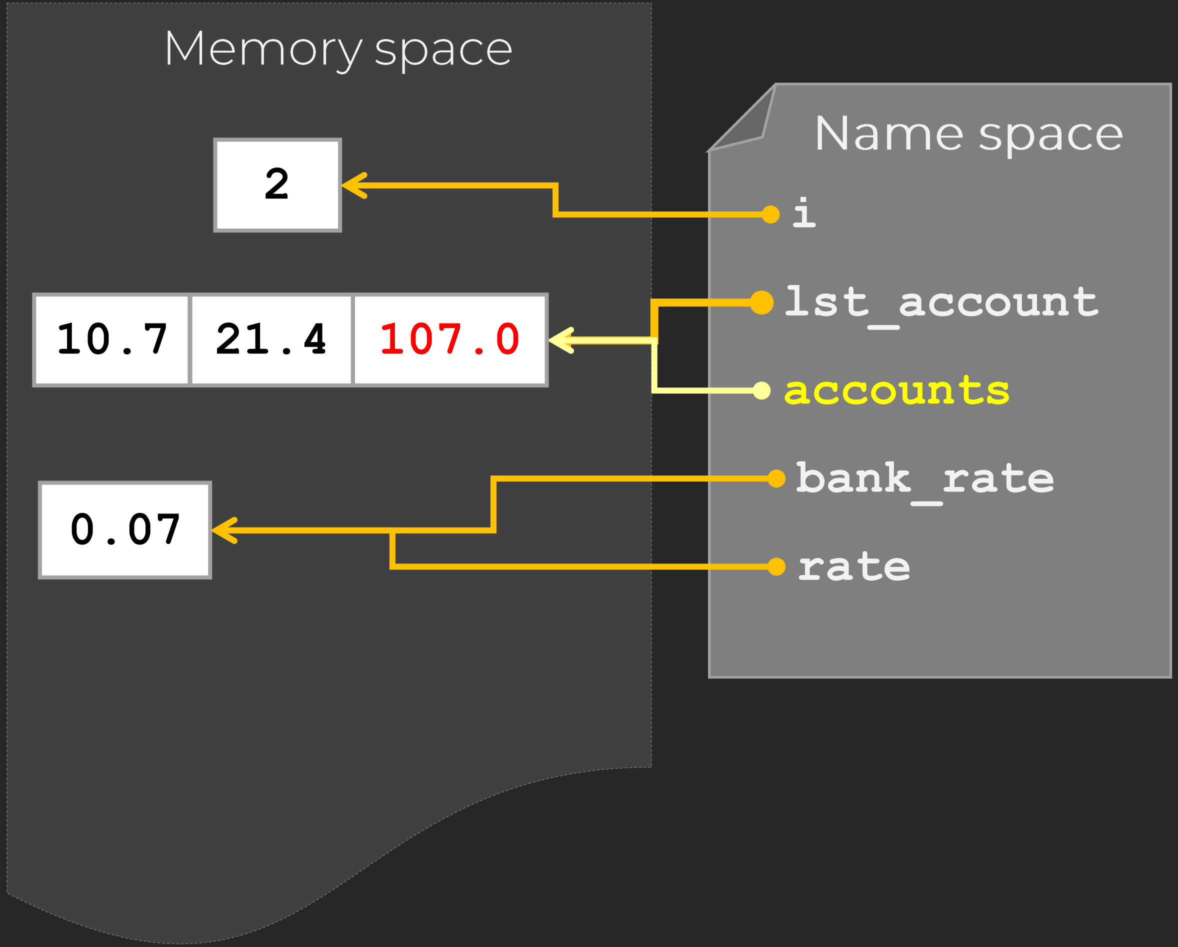
Python shell



Code

```
def addInterestAll(accounts, rate):
    for i in range(len(accounts)):
        accounts[i] *= (1+rate)

lst_account = [10.0, 20.0, 100.0]
bank_rate = 0.07
addInterestAll(lst_accounts, bank_rate)
print("new accounts balance:",\
      lst_accounts)
```

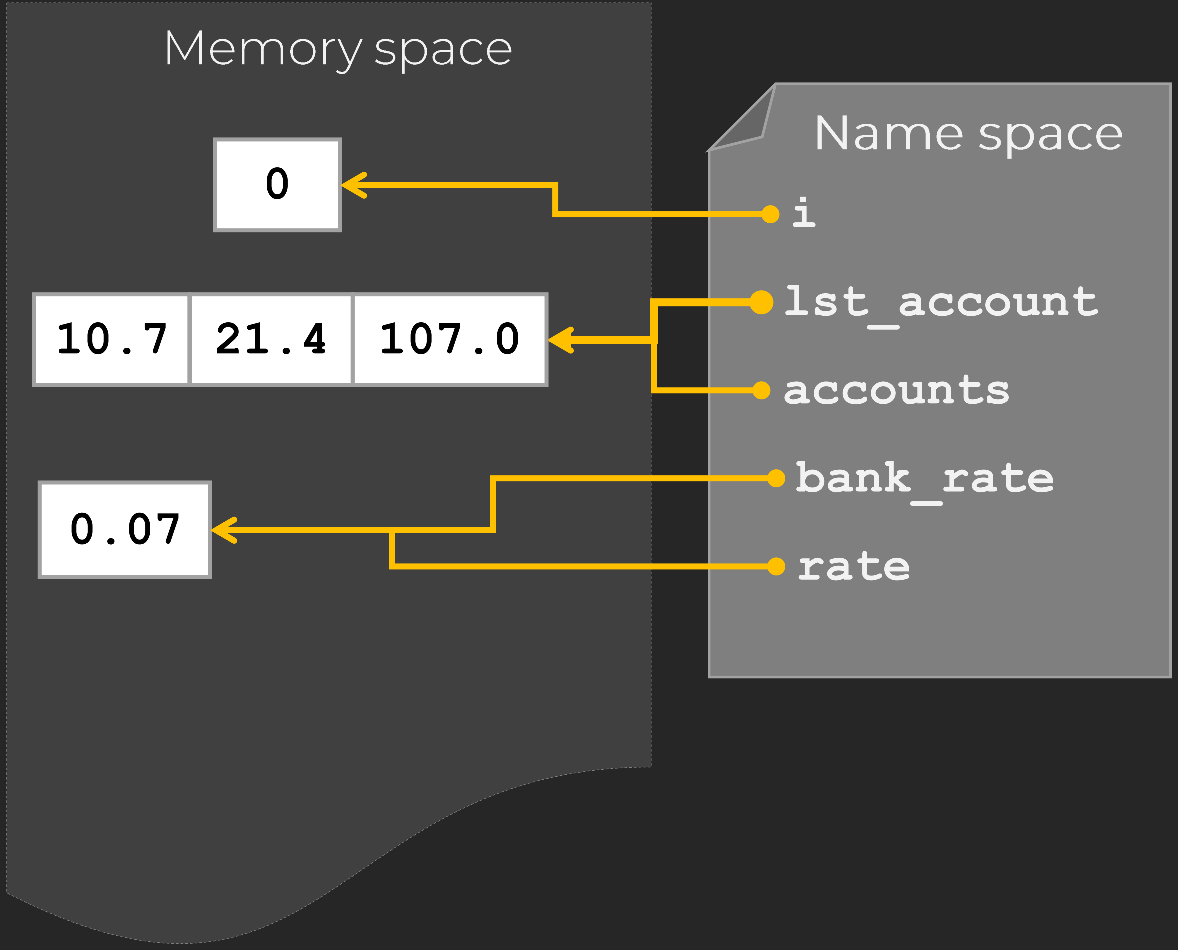
Python shell



Code

```
def addInterestAll(accounts, rate):
    for i in range(len(accounts)):
        accounts[i] *= (1+rate)

lst_account = [10.0, 20.0, 100.0]
bank_rate = 0.07
addInterestAll(lst_accounts, bank_rate)
print("new accounts balance:",\
      lst_accounts)
```

Python shell

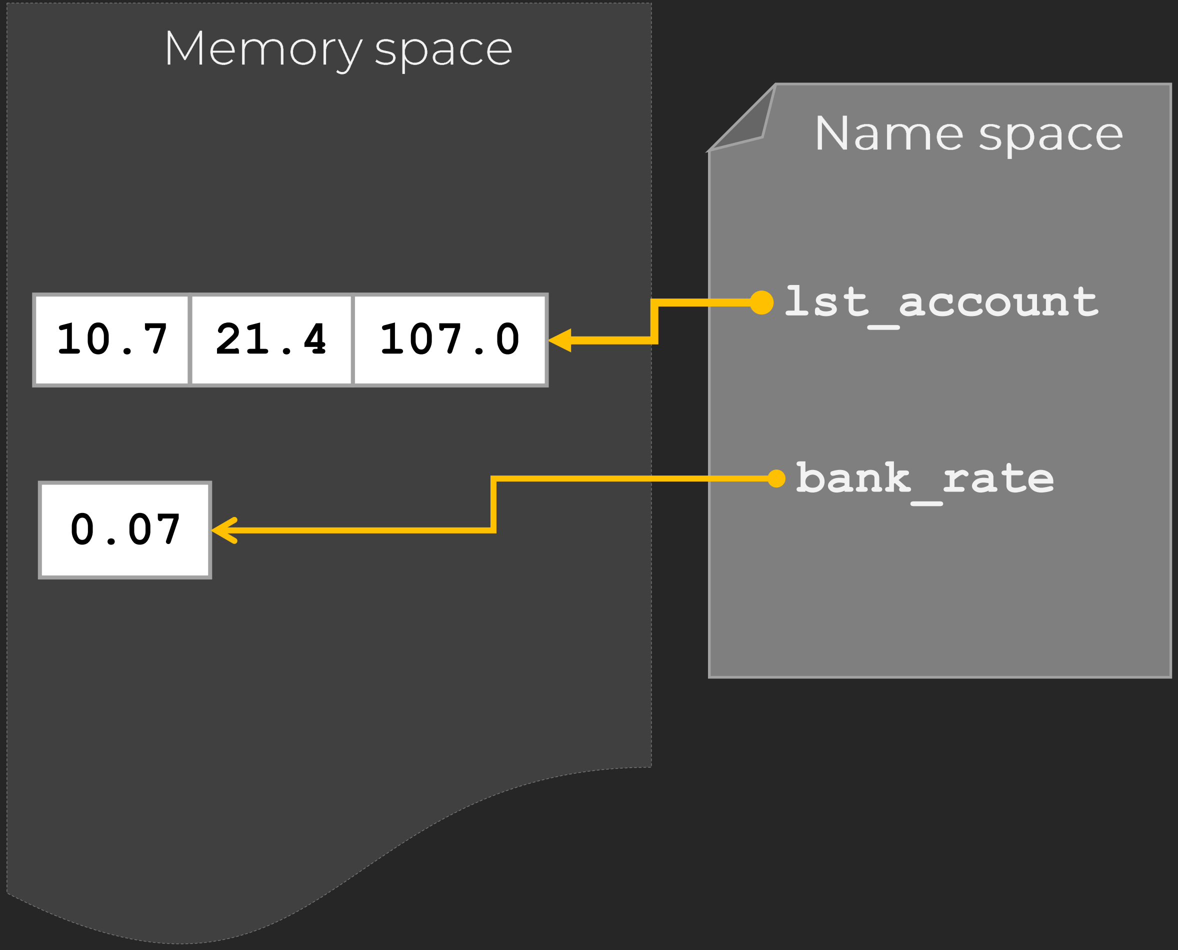


Code

```
def addInterestAll(accounts, rate):
    for i in range(len(accounts)):
        accounts[i] *= (1+rate)

lst_account = [10.0, 20.0, 100.0]
bank_rate = 0.07
addInterestAll(lst_accounts, bank_rate)
print("new accounts balance:",\
      lst_accounts)
```

Python shell

```
new accounts balance:[10.7,21.4,107.0]
```



“

Do not modify mutable arguments in a function unless the caller expect it.

”

If must be explicit in the documentation provided with the function, that is in the docstring.

You have seen how to call a function using positional and keyword arguments. We also learn to be cautious when passing a mutable object in a function parameter as side effects can occur.

There is one more important thing to look at, the scope of variable.