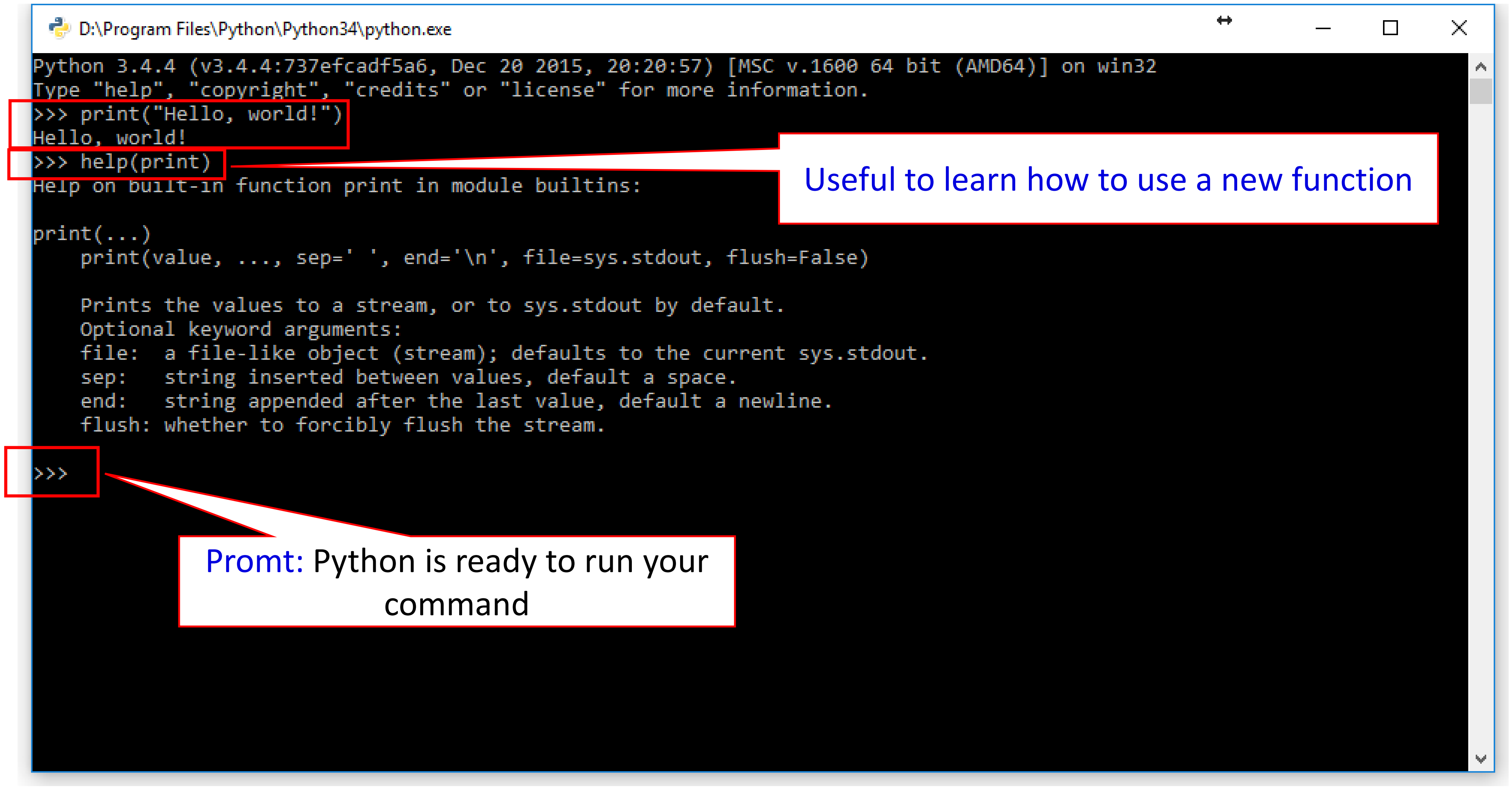
➤ Run Python Interpreter

- Find python.exe in the installed files



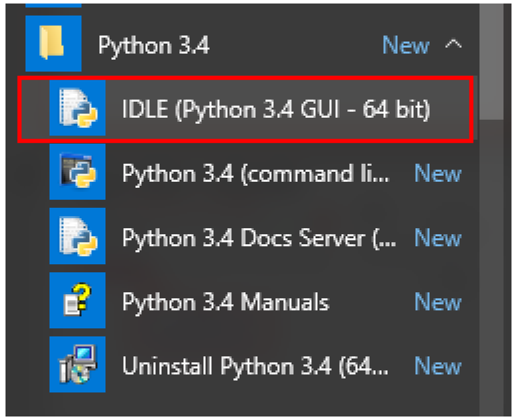
```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, world!")
Hello, world!
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>>
```

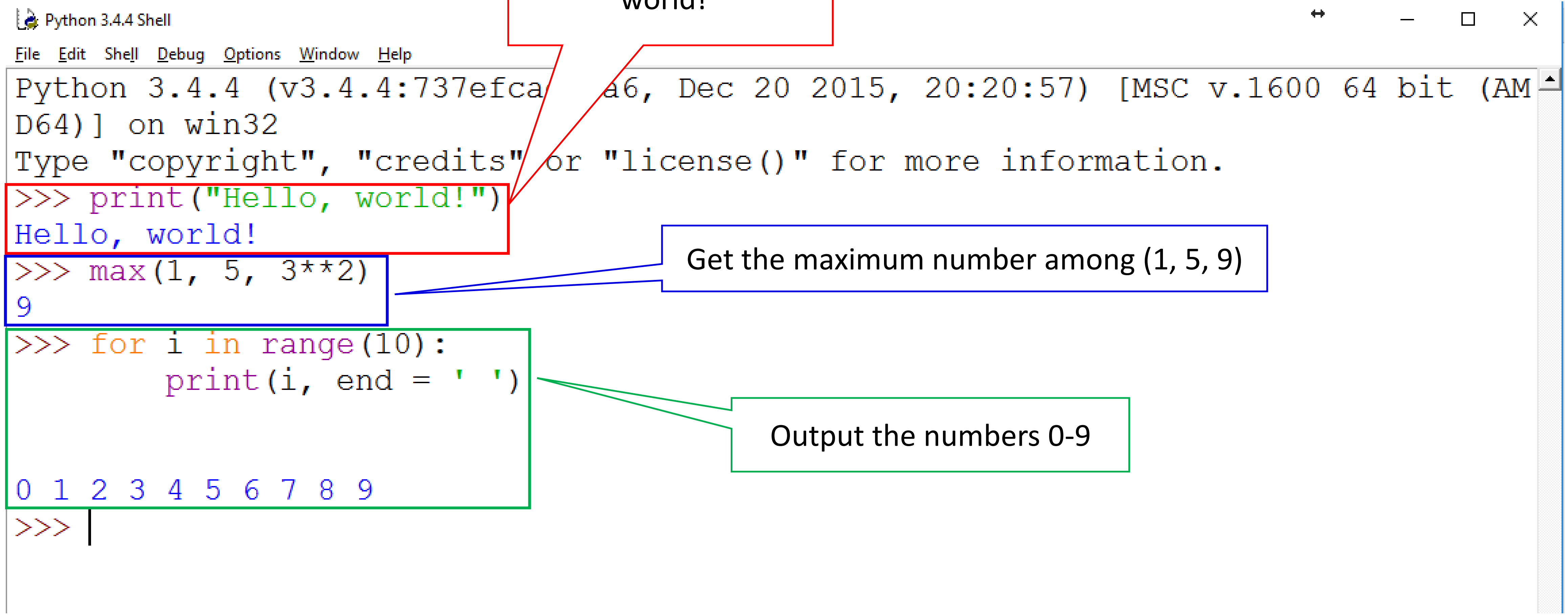
Useful to learn how to use a new function

Promt: Python is ready to run your command

➢ Run Python Shell (IDLE)

- Similar to Python interpreter


Python 3.4
New
IDLE (Python 3.4 GUI - 64 bit)
Python 3.4 (command li... New
Python 3.4 Docs Server (... New
Python 3.4 Manuals New
Uninstall Python 3.4 (64... New



```
Python 3.4.4 (v3.4.4:737efca  a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello, world!")
Hello, world!
>>> max(1, 5, 3**2)
9
>>> for i in range(10):
        print(i, end = ' ')


0 1 2 3 4 5 6 7 8 9
>>> 
```

Output the “Hello, world!”

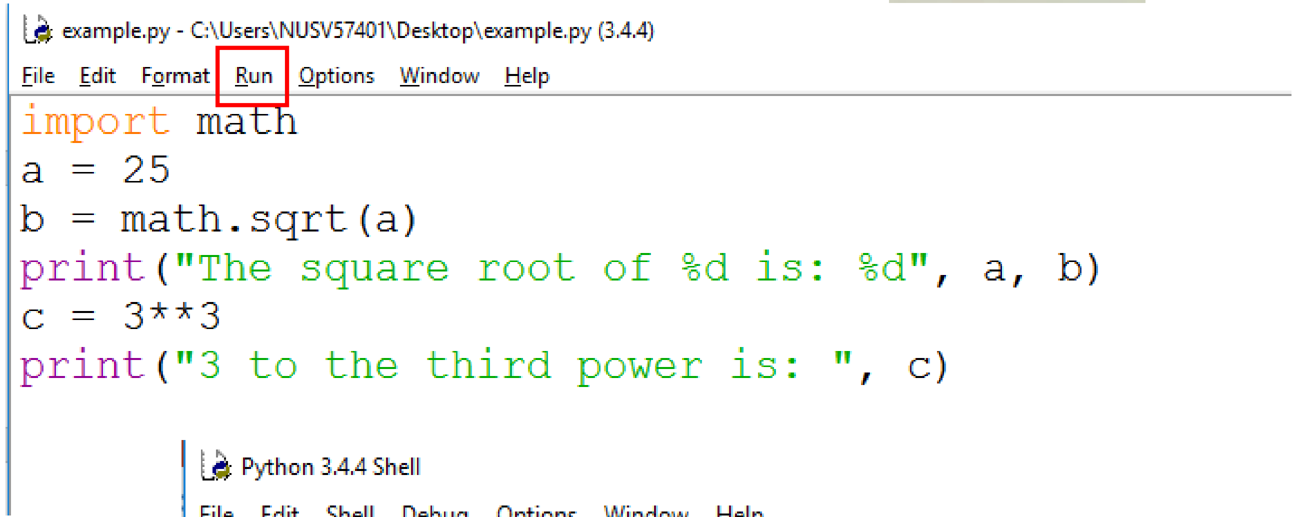
Get the maximum number among (1, 5, 9)

Output the numbers 0-9

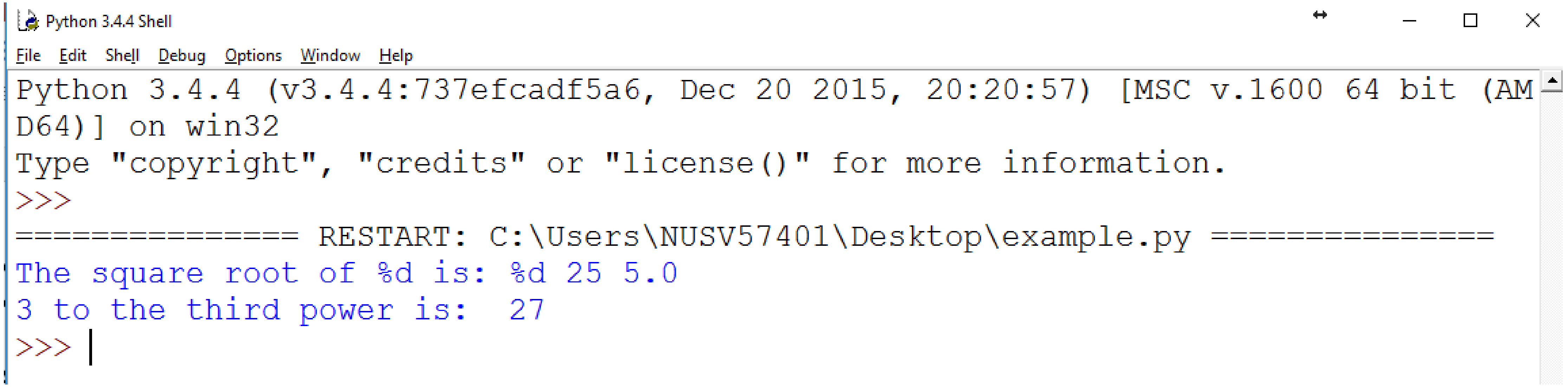
➢ Program file and Run

- Create file *.py and Edit

  useful for long program


example.py - C:\Users\NUSV57401\Desktop\example.py (3.4.4)
File Edit Format Run Options Window Help

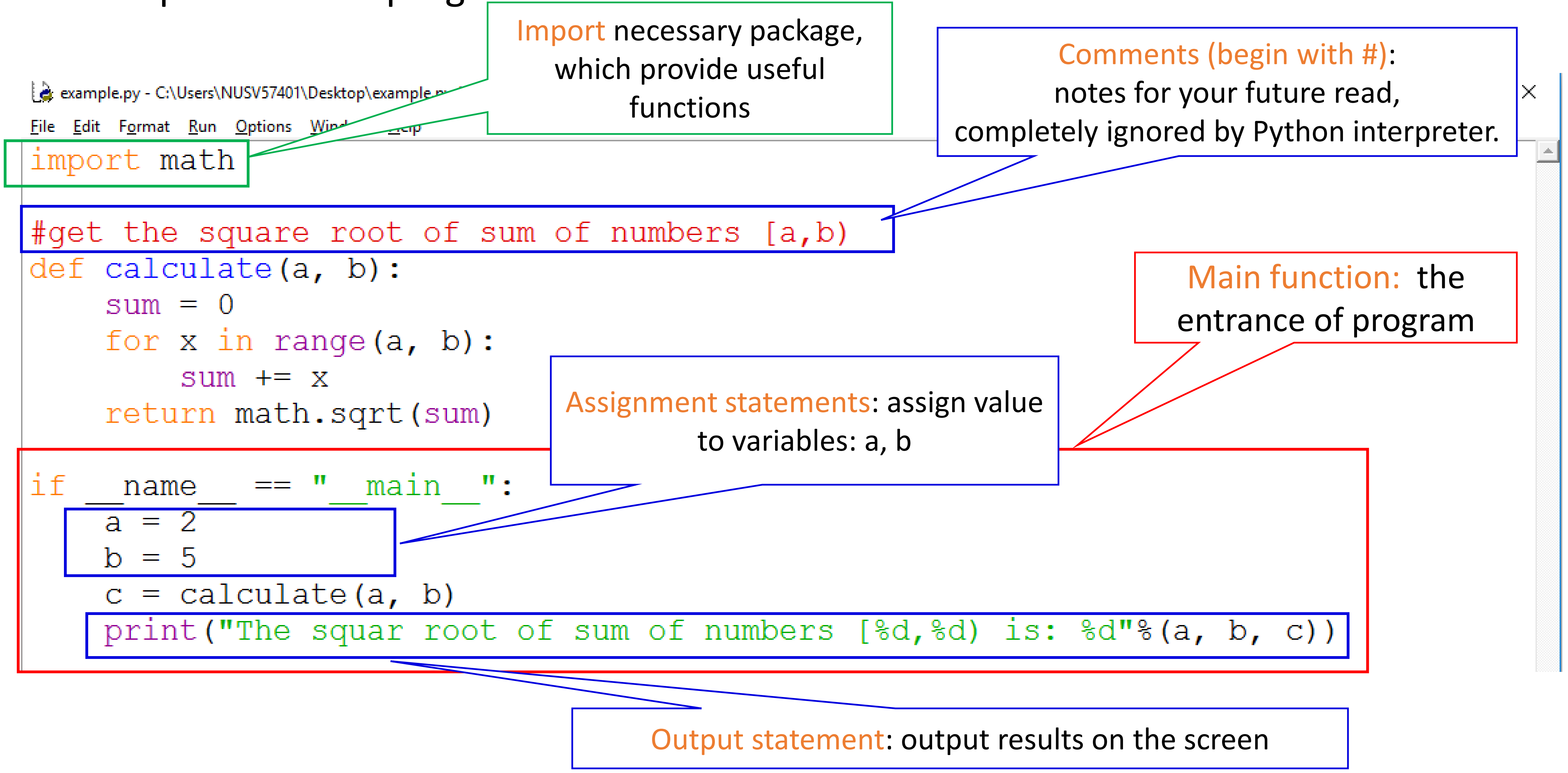
```
import math
a = 25
b = math.sqrt(a)
print("The square root of %d is: %d", a, b)
c = 3**3
print("3 to the third power is: ", c)
```



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
=============== RESTART: C:\Users\NUSV57401\Desktop\example.py ===============
The square root of %d is: %d 25 5.0
3 to the third power is:  27
>>> |
```

➤ Program

- Components of a program


example.py - C:\Users\NUSV57401\Desktop\example.p
File Edit Format Run Options Win... ...elp

```
import math

#get the square root of sum of numbers [a,b)
def calculate(a, b):
    sum = 0
    for x in range(a, b):
        sum += x
    return math.sqrt(sum)

if __name__ == "__main__":
    a = 2
    b = 5
    c = calculate(a, b)
    print("The squar root of sum of numbers [%d,%d) is: %d"%(a, b, c))
```

Import necessary package, which provide useful functions

Comments (begin with #): notes for your future read, completely ignored by Python interpreter.

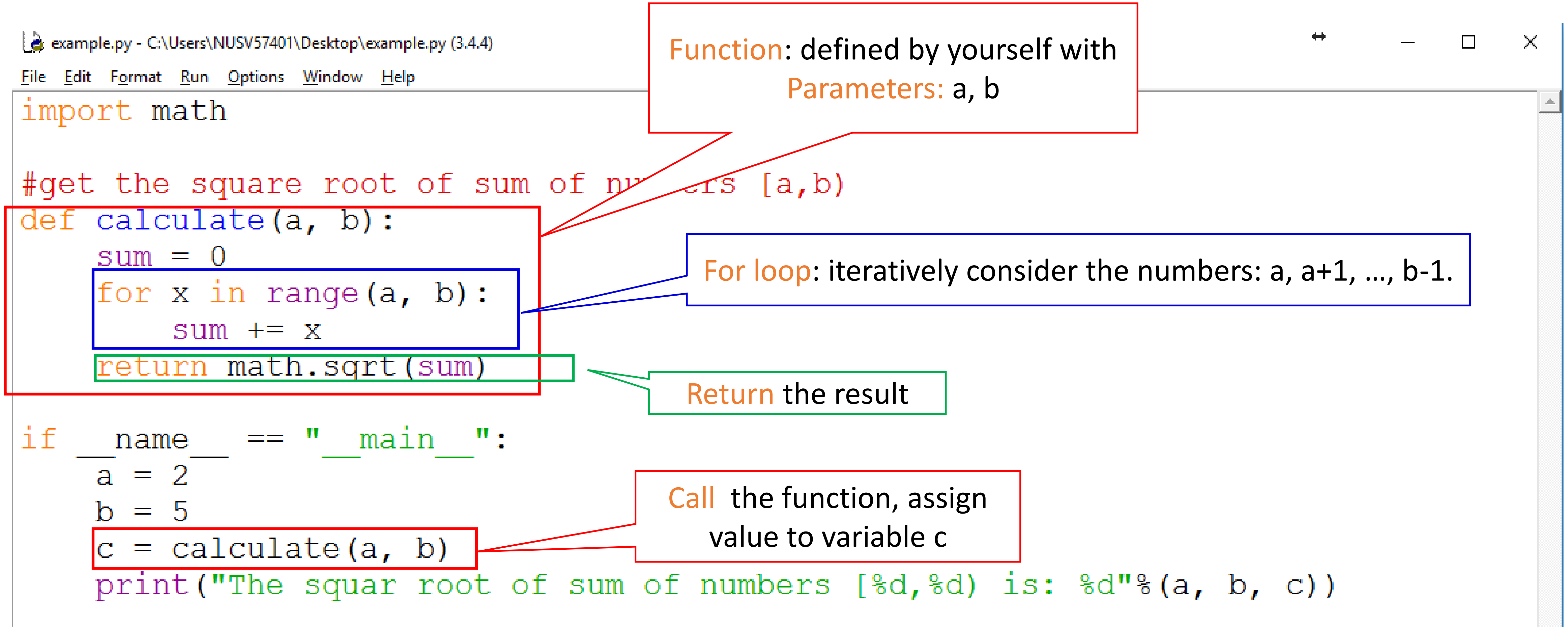
Main function: the entrance of program

Assignment statements: assign value to variables: a, b

Output statement: output results on the screen

Program

- Components of a program
- Notice: Python is very strict about indentation, like in for Loop



example.py - C:\Users\NUSV57401\Desktop\example.py (3.4.4)
File Edit Format Run Options Window Help

```
import math

#get the square root of sum of numbers [a,b)
def calculate(a, b):
    sum = 0
    for x in range(a, b):
        sum += x
    return math.sqrt(sum)

if __name__ == "__main__":
    a = 2
    b = 5
    c = calculate(a, b)
    print("The squar root of sum of numbers [%d,%d) is: %d"%(a, b, c))
```

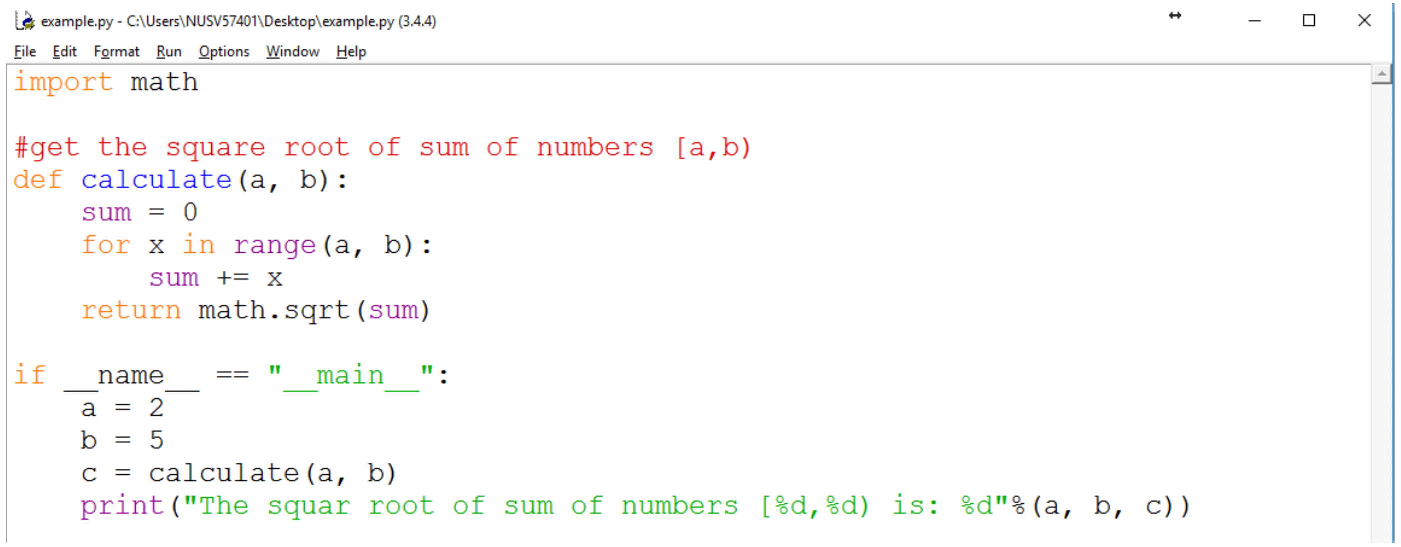
Function: defined by yourself with Parameters: a, b

For loop: iteratively consider the numbers: a, a+1, ..., b-1.

Return the result

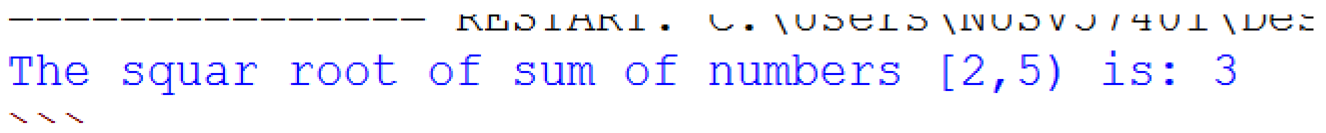
Call the function, assign value to variable c

➢ Program

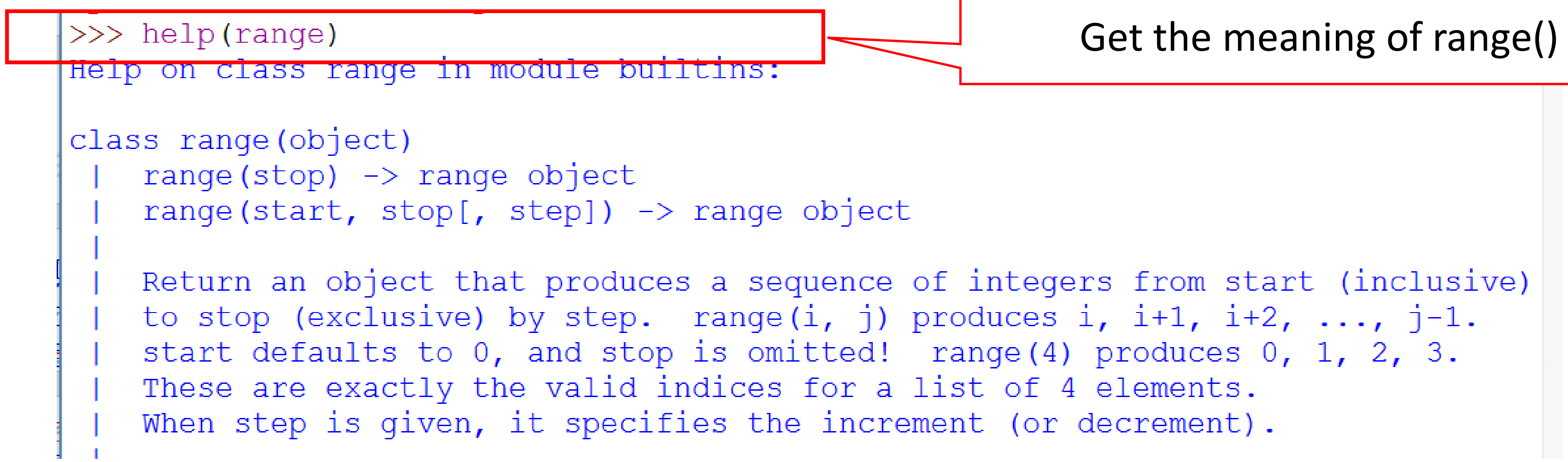

example.py - C:\Users\NUSV57401\Desktop\example.py (3.4.4)
File Edit Format Run Options Window Help

```
import math

#get the square root of sum of numbers [a,b)
def calculate(a, b):
    sum = 0
    for x in range(a, b):
        sum += x
    return math.sqrt(sum)

if __name__ == "__main__":
    a = 2
    b = 5
    c = calculate(a, b)
    print("The squar root of sum of numbers [%d,%d) is: %d"%(a, b, c))
```

Run, get



```
================ RESTART: C:\Users\NUSV57401\Des
The squar root of sum of numbers [2,5) is: 3
>>>
```



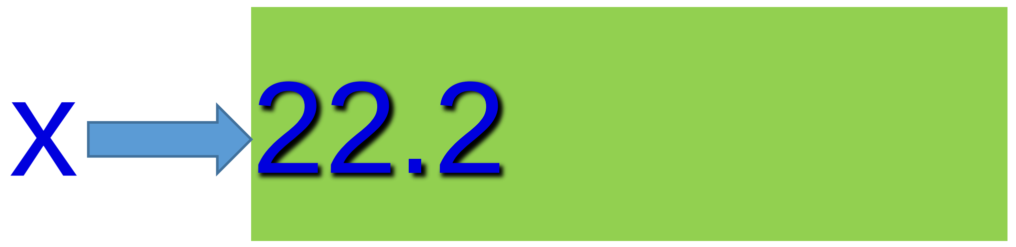
```
>>> help(range)
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start (inclusive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2, ..., j-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0, 1, 2, 3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |
```

Get the meaning of range()

➢ Variable

- is a named place in the memory, where you can store and retrieve data
- The content can be changed

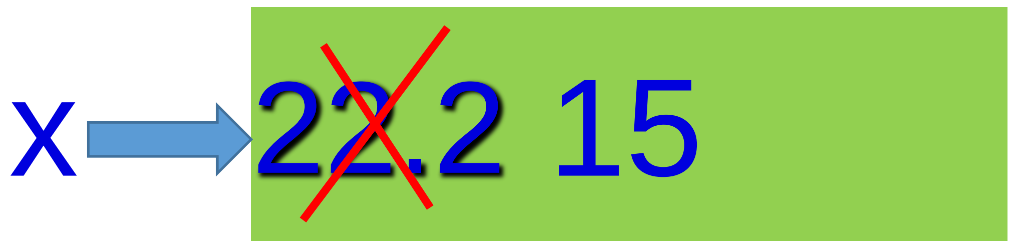
x = 22.2



Variable

- is a named place in the memory, where you can store and retrieve data
- The content can be changed

```
x = 22.2
x = 15
```

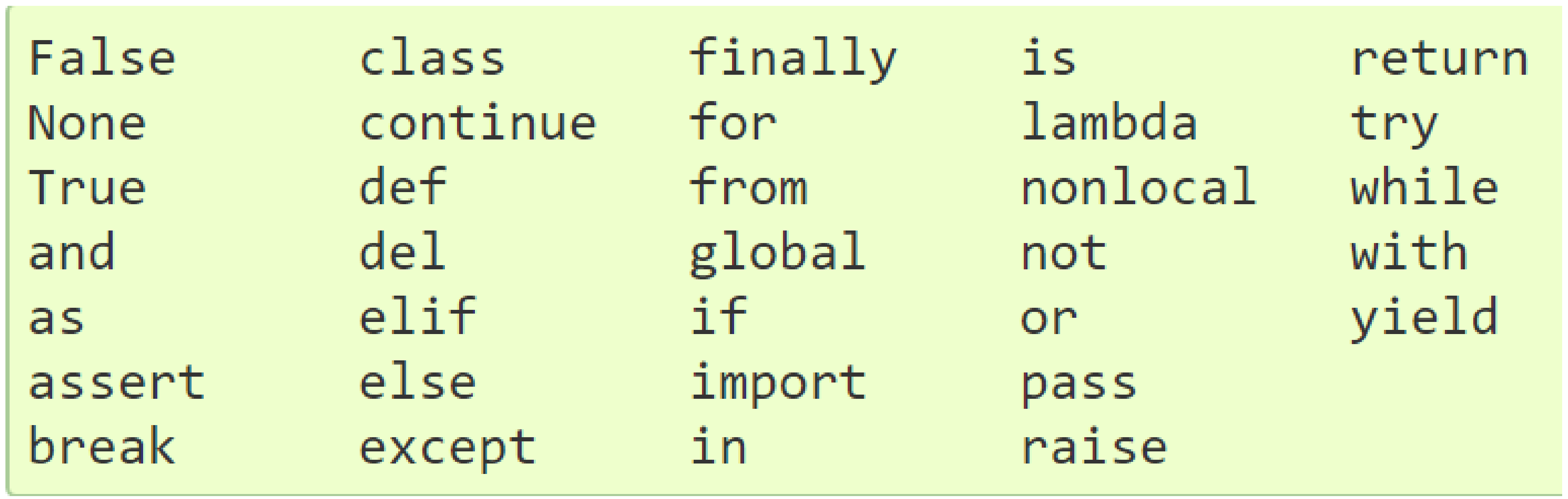


- Name Rules:
  - Consist of letters, numbers, and underscores(_)
  - Must start with a letter or underscore
  - Case sensitive
  - Cannot use the reserved words by Python

Right: student, Singapore, fire32, _name
Wrong: 3pigs, #type, age.18

➢ Variable---Reserved Words

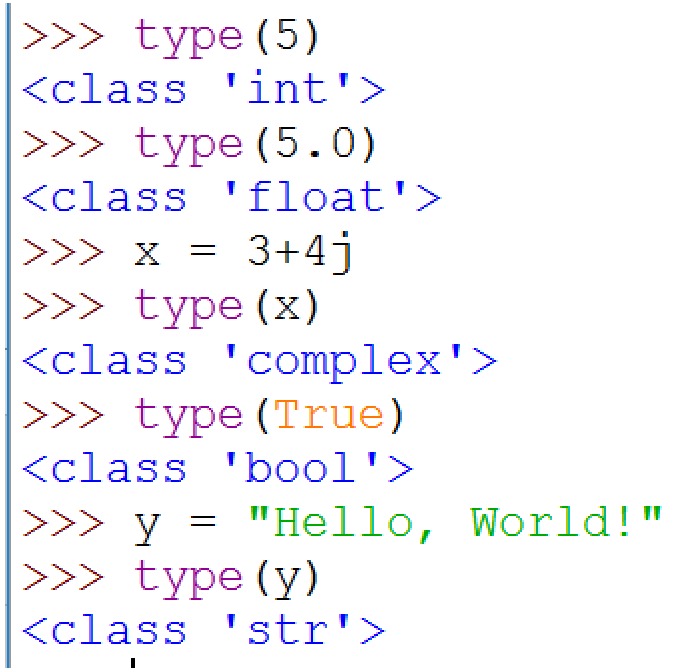
- Used by Python



```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

➢ Variable Types

- No need to specify explicitly, Python dynamically decide the type of variables
- Possible types:
  - Numeric: int (Integer), float (Floating point), complex (Imaginary)
  - Boolean: True, False
  - String: using single-quotes(‘) or double-quotes(“)

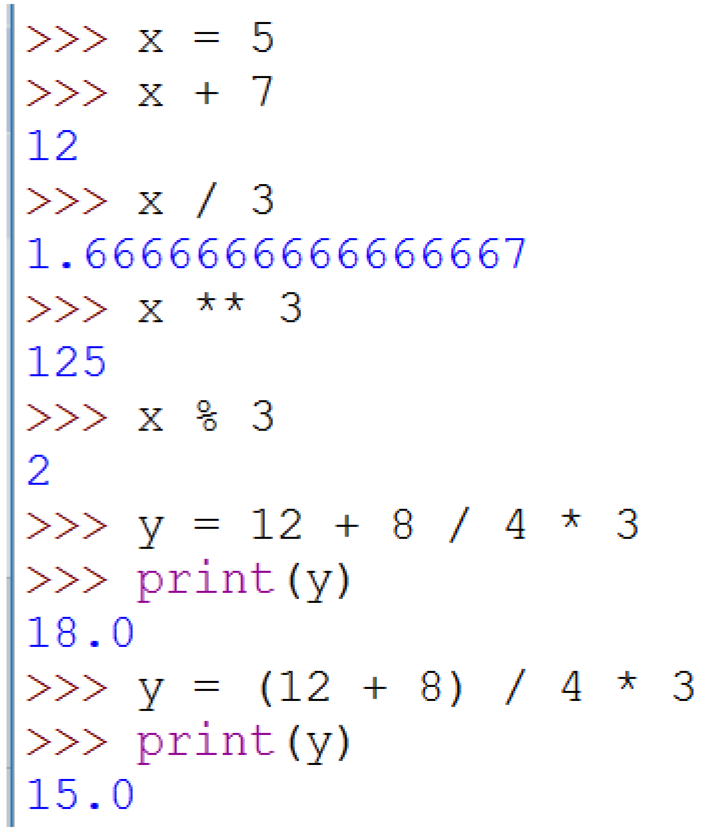


```
>>> type(5)
<class 'int'>
>>> type(5.0)
<class 'float'>
>>> x = 3+4j
>>> type(x)
<class 'complex'>
>>> type(True)
<class 'bool'>
>>> y = "Hello, World!"
>>> type(y)
<class 'str'>
```

Numeric Expressions

- Use parenthesis() to give high operator precedence

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder



```
>>> x = 5
>>> x + 7
12
>>> x / 3
1.6666666666666667
>>> x ** 3
125
>>> x % 3
2
>>> y = 12 + 8 / 4 * 3
>>> print(y)
18.0
>>> y = (12 + 8) / 4 * 3
>>> print(y)
15.0
```

Boolean Expressions

Operation	Result
and	Either one is False gets False
or	Either one is True gets True
not	The opposite value

- Other operation

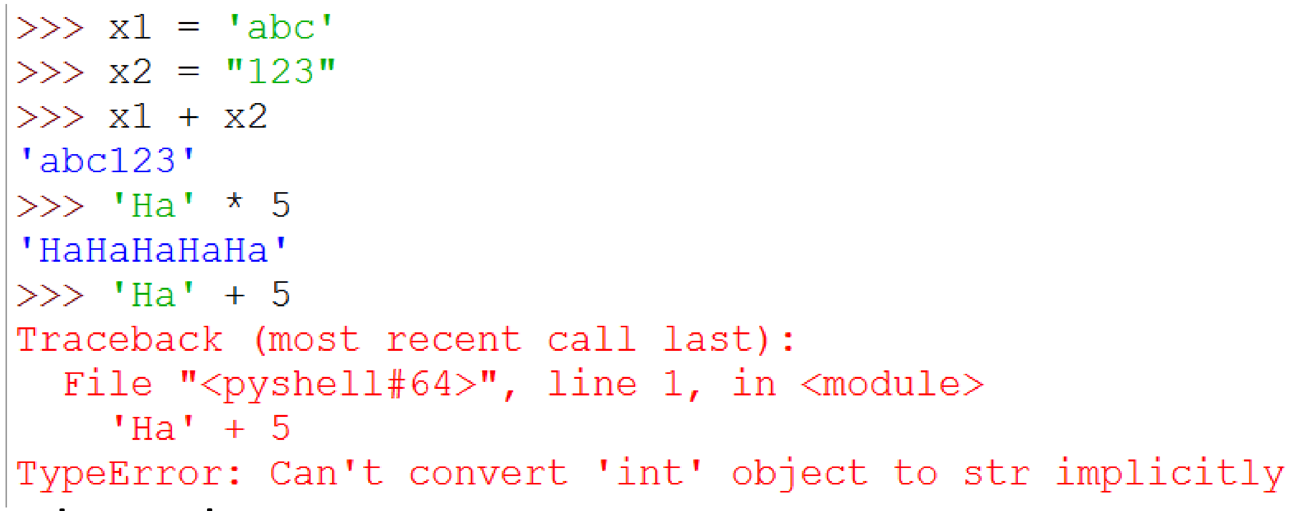
Operation	Meaning
in	Is in set?
not in	Is not in set?

```
>>> True and True
True
>>> True and False
False
>>> True or False
True
>>> False or False
False
>>> not True
False
>>> not False
True
>>> a = {0, 2, 4, 5}
>>> 1 in a
False
>>> 5 in a
True
>>> 5 not in a
False
```

String Expressions

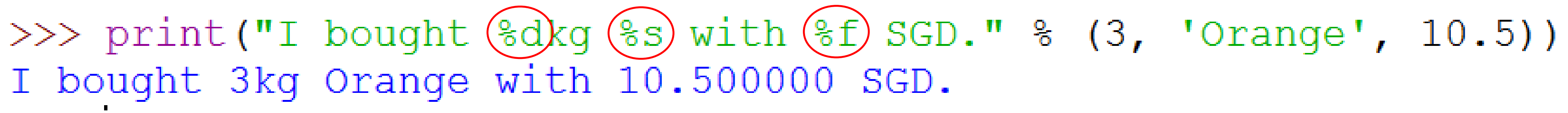
- Use single-quotes(‘) or double-quotes(“)
- Some operators apply to strings

Operator	Operation
+	Concatenation
*	Multiple concatenation



```
>>> x1 = 'abc'
>>> x2 = "123"
>>> x1 + x2
'abc123'
>>> 'Ha' * 5
'HaHaHaHaHa'
>>> 'Ha' + 5
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    'Ha' + 5
TypeError: Can't convert 'int' object to str implicitly
```

- + can not be used between string and number
- Print-style string formatting *format % values*, useful to output information
  - % is the string formatting operator, % conversion specifications in format are replaced with elements in *values*
  - *format* is a string, *values* must be a tuple with exactly the number of items specified by the *format*



```
>>> print("I bought %dkg %s with %f SGD." % (3, 'Orange', 10.5))
I bought 3kg Orange with 10.500000 SGD.
```

➢ Print-style String Format

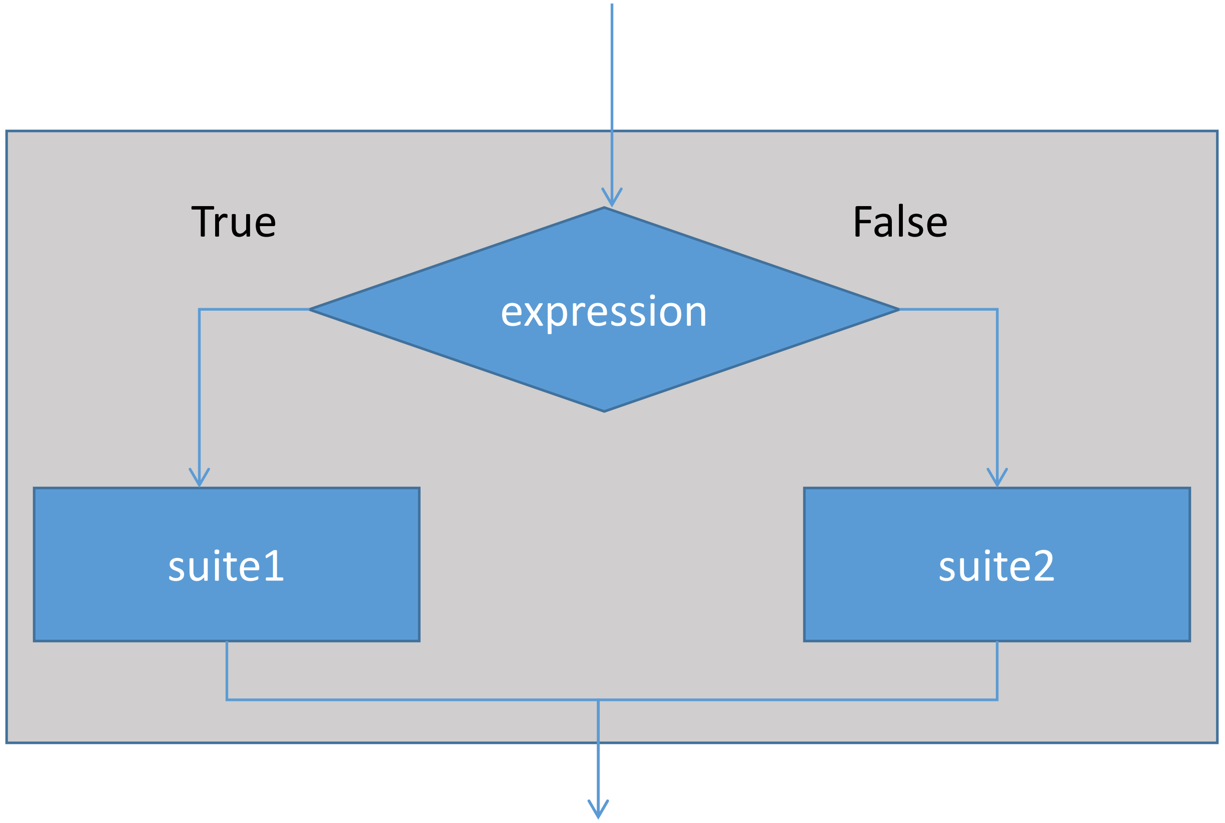
- The conversion types are:

Conversion	Meaning
%s	String (converts any Python object using str())
%a	String (converts any Python object using ASCII())
%c	Single character (accepts integer or single character string)
%d	Signed integer decimal number
%f	Floating point decimal format
%e	Floating point exponential format (lower case)
%E	Floating point exponential format (upper case)
%o	Signed octal value
%x	Signed hexadecimal value

➢ Conditional Execution---*if* statement

- The program is usually executed in sequence
- *if* statement is used for conditional execution
- Usage:

```
if expression:
    suite1
(elif expression:
    suitei)*
[else:
    suite2]
```



➢ Conditional Execution---*if* statement

```
if expression:
    suite1
(elif expression:
    suitei)*
[else:
    suite2]
```

- if statement can be used without else clause
- Multi-way can also be used

```
>>> def test1(grade):
        if grade >= 85:
                print("Good")


>>> test1(90)
Good
>>> test1(70)
>>>
```

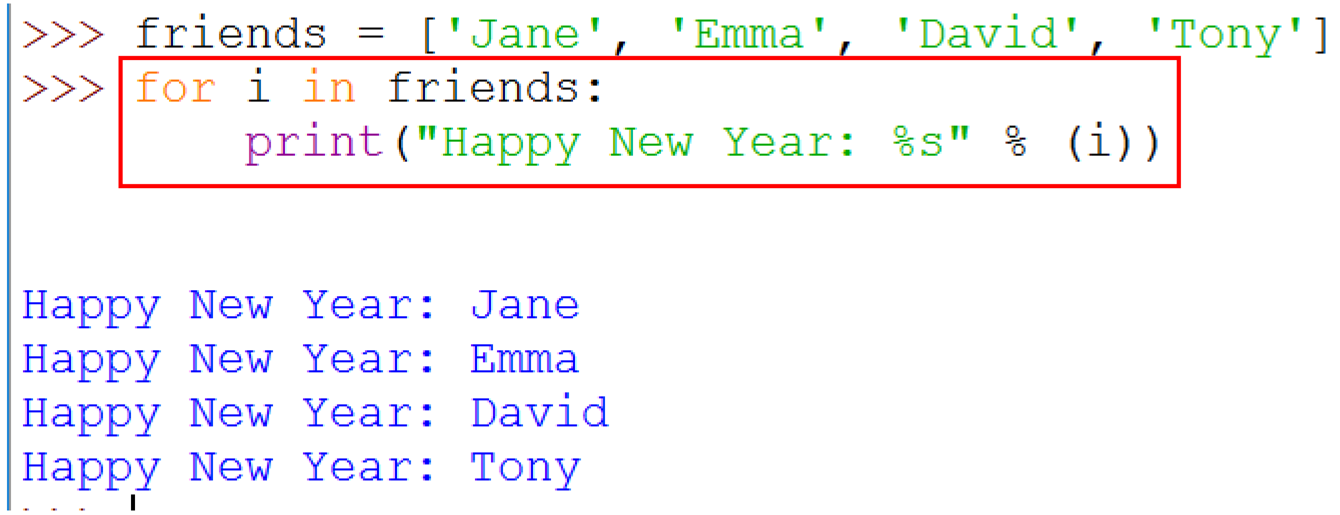
```
>>> def test2(grade):
        if grade >= 85:
                print("Good")
        else:
                print("Not good")


>>> test2(90)
Good
>>> test2(70)
Not good
```

```
>>> def test3(grade):
        if grade >= 85:
                print("Excellent")
        elif grade >= 70:
                print("Good")
        elif grade >= 60:
                print("Pass")
        else:
                print("Failed")


>>> test3(90)
Excellent
>>> test3(80)
Good
>>> test3(65)
Pass
>>> test3(50)
Failed
```

➢ Loop Execution---*for* statement

- *for* statement is used to iterate over a list of elements
- Usage:

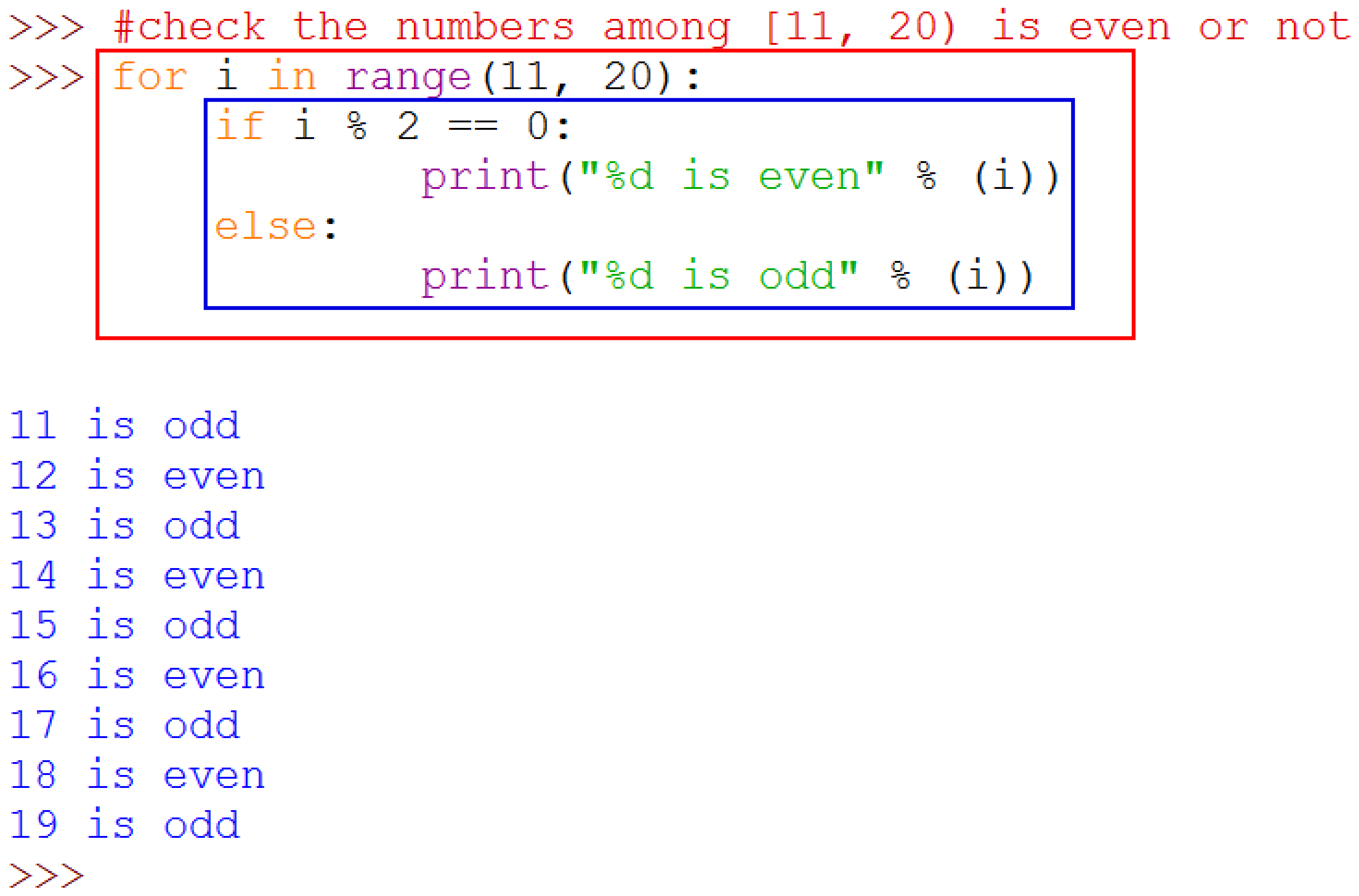
```
for item in expression_list:
    suite
```



```
>>> friends = ['Jane', 'Emma', 'David', 'Tony']
>>> for i in friends:
        print("Happy New Year: %s" % (i))


Happy New Year: Jane
Happy New Year: Emma
Happy New Year: David
Happy New Year: Tony
```

Loop Execution---*for* statement

- *for* statement can be used with *if* statement to do complex things

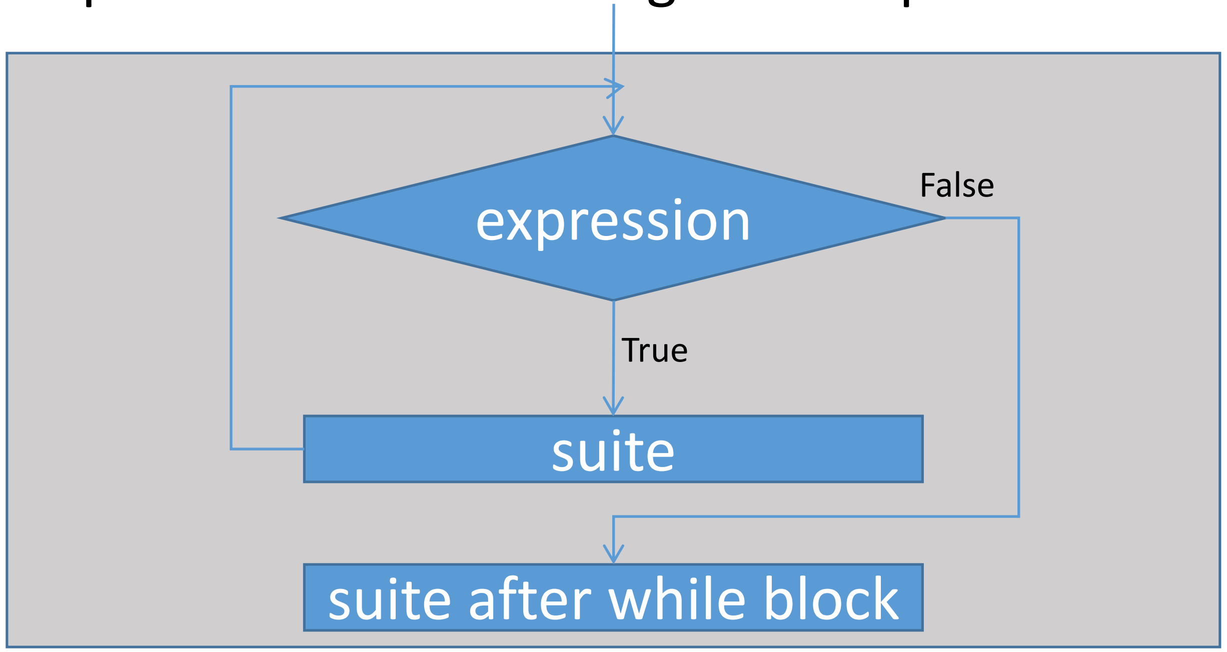


```
>>> #check the numbers among [11, 20) is even or not
>>> for i in range(11, 20):
        if i % 2 == 0:
                print("%d is even" % (i))
        else:
                print("%d is odd" % (i))


11 is odd
12 is even
13 is odd
14 is even
15 is odd
16 is even
17 is odd
18 is even
19 is odd
>>>
```

Loop Execution---*while* statement

- *while* statement is used to repeat execution as long as an expression is true
- Usage:

```
while expression:
    suite
```



- Example and Result

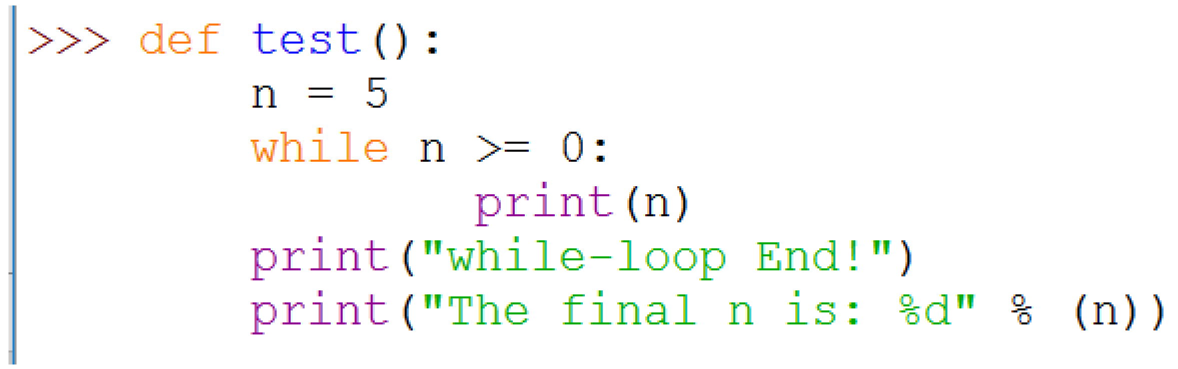
```
>>> def test():
        n = 5
        while n >= 0:
                print(n)
                n = n - 1
        print("while-loop End!")
        print("The final n is: %d" % (n))
```

```
>>> test()
5
4
3
2
1
0
while-loop End!
The final n is: -1
```

➢ Loop Execution---*while* statement

- Notice: avoid the infinite loop
- Expression must be false after certain number of iterations

```
while expression:
    suite
```



```
>>> def test():
        n = 5
        while n >= 0:
                print(n)
        print("while-loop End!")
        print("The final n is: %d" % (n))
```

What is the result of this loop?

Loop Execution---*break* and *continue*

- *break* statement terminates the current loop, and jumps to the statements immediately after the loop
- *continue* statement skips the rest of suite, and jumps to the top of the loop, and starts the next iteration

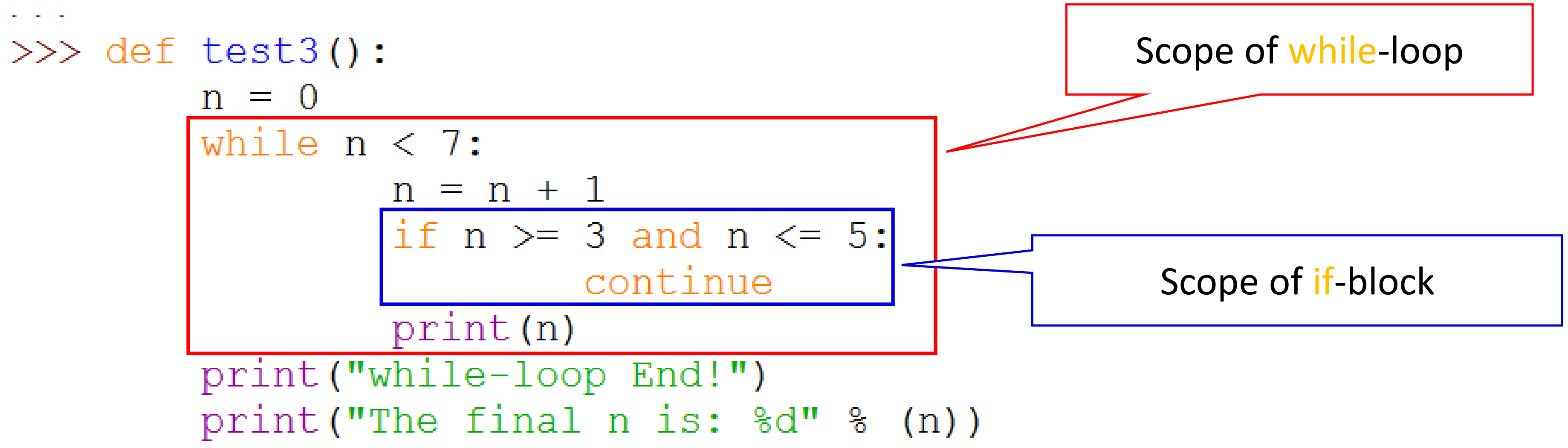
```
>>> def test1():
        n = 0
        while n < 7:
                n = n + 1
                print(n)
        print("while-loop End!")
        print("The final n is: %d" % (n))


>>> test1()
1
2
3
4
5
6
7
while-loop End!
The final n is: 7
```

```
>>> def test2():
        n = 0
        while n < 7:
                n = n + 1
                if n >= 3 and n <= 5:
                        break
                print(n)
        print("while-loop End!")
        print("The final n is: %d" % (n))


>>> test2()
1
2
while-loop End!
The final n is: 3
```

```
>>> def test3():
        n = 0
        while n < 7:
                n = n + 1
                if n >= 3 and n <= 5:
                        continue
                print(n)
        print("while-loop End!")
        print("The final n is: %d" % (n))


>>> test3()
1
2
6
7
while-loop End!
The final n is: 7
```

➤ Indentation

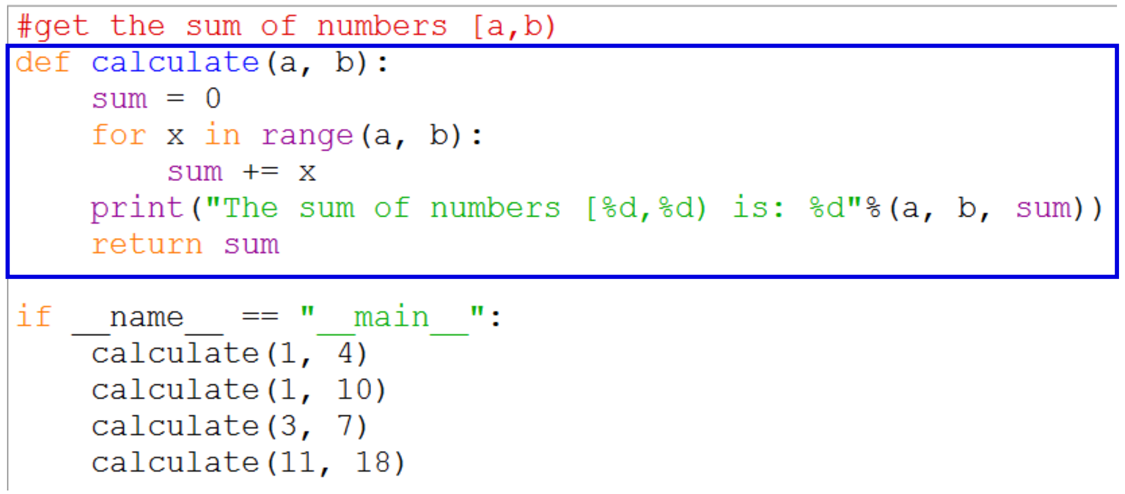
- Python is very strict about indentation
- Indentation is used to indicate the scope of the block (if/for/while)
- Increase indent after an if/for/while statement (after :)
- Maintain indent in the scope of the same block
- Reduce indent after the end of a block
- Blank lines are ignored – they do not affect indentation
- Comments on a line are also ignored



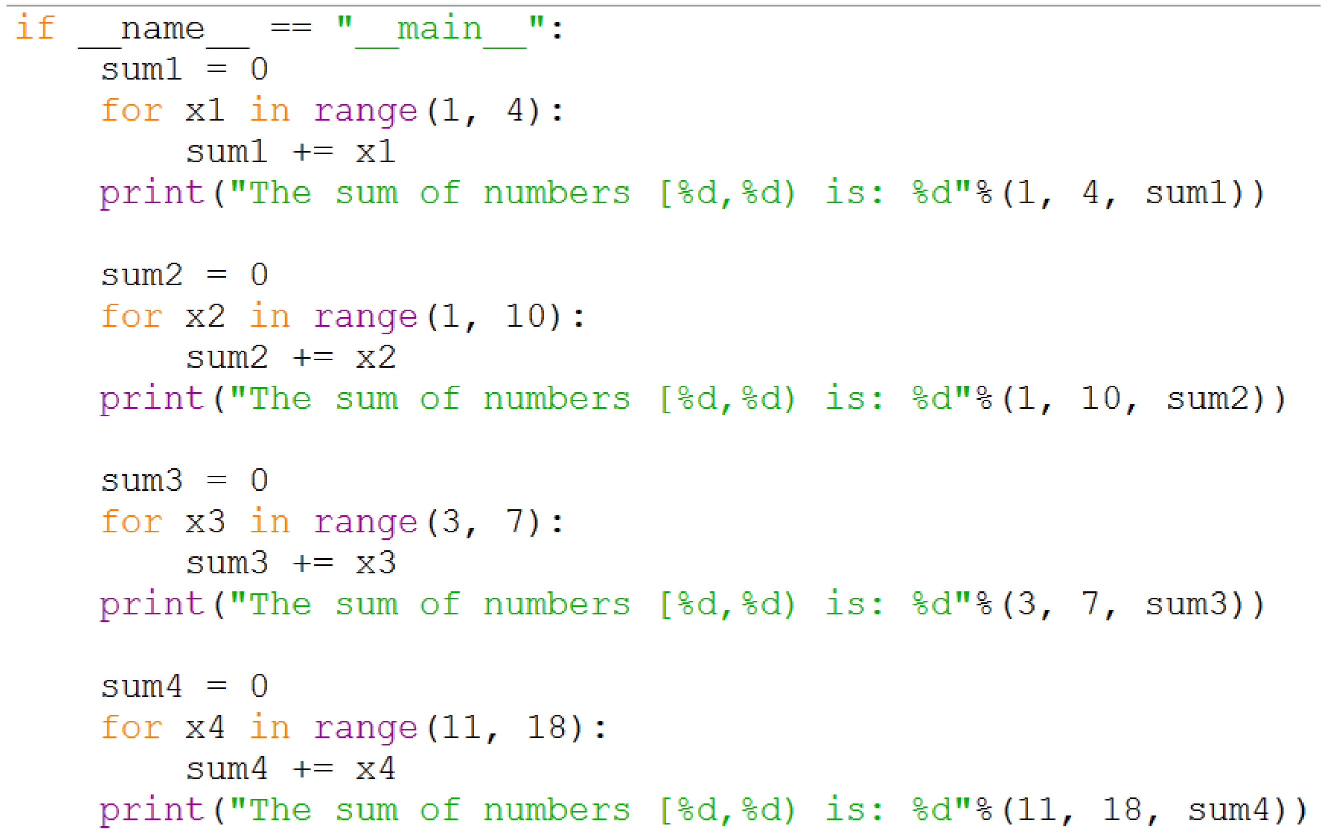
```
...
>>> def test3():
        n = 0
        while n < 7:
                n = n + 1
                if n >= 3 and n <= 5:
                        continue
                print(n)
        print("while-loop End!")
        print("The final n is: %d" % (n))
```

➢ Function

- Part of program may be repeated more than once, then it can be wrote as a function.
- Advantage: it is easy to read or modify the code, only one place
- Function likes a black box which takes in source data, processes it, and passes it back.



```python
#get the sum of numbers [a,b)
def calculate(a, b):
    sum = 0
    for x in range(a, b):
        sum += x
    print("The sum of numbers [%d,%d) is: %d"%(a, b, sum))
    return sum

if __name__ == "__main__":
    calculate(1, 4)
    calculate(1, 10)
    calculate(3, 7)
    calculate(11, 18)
```



```python
if __name__ == "__main__":
    sum1 = 0
    for x1 in range(1, 4):
        sum1 += x1
    print("The sum of numbers [%d,%d) is: %d"%(1, 4, sum1))

    sum2 = 0
    for x2 in range(1, 10):
        sum2 += x2
    print("The sum of numbers [%d,%d) is: %d"%(1, 10, sum2))

    sum3 = 0
    for x3 in range(3, 7):
        sum3 += x3
    print("The sum of numbers [%d,%d) is: %d"%(3, 7, sum3))

    sum4 = 0
    for x4 in range(11, 18):
        sum4 += x4
    print("The sum of numbers [%d,%d) is: %d"%(11, 18, sum4))
```

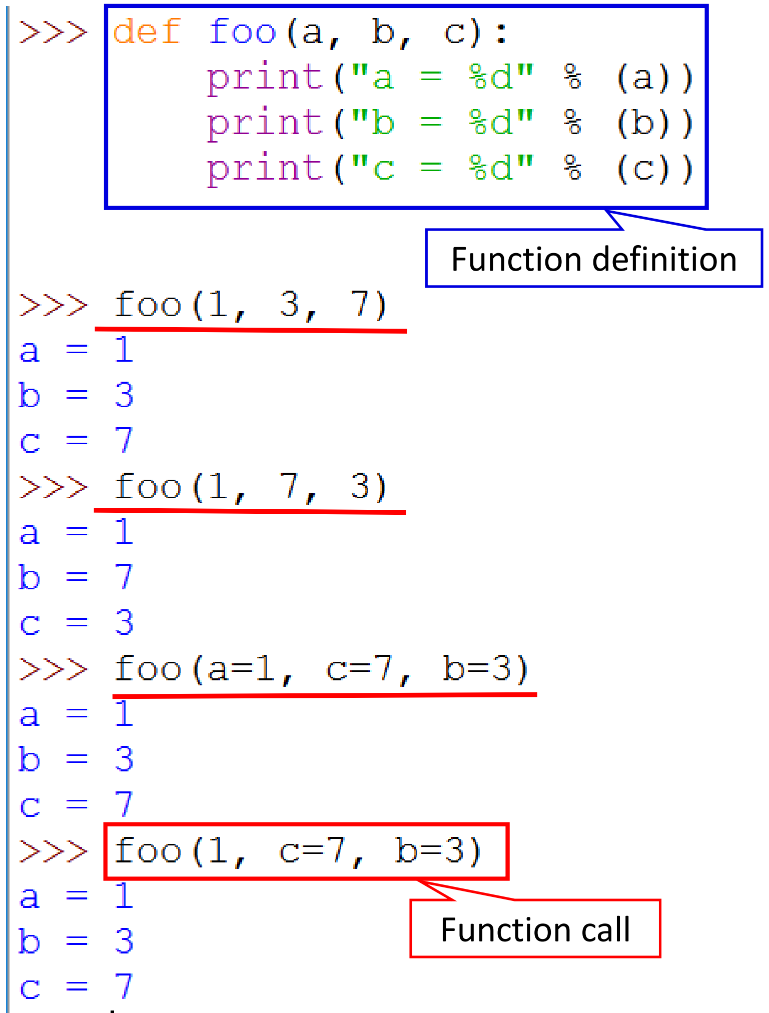
➢ Function

- Function definition:
  - Function name rules are similar to variables
  - Parameters: data supplied to the function. They can be empty or more than one (delimited with ,)
  - The result of the function is output via return

```
def funcname([parameter_list]):
    suite
    [return result]
```

- Function call:
  - A function is executed by calling the function
  - Arguments: values passed to a function when calling the function.
    - ✓ Positional arguments: pass values in the sequence of parameters
    - ✓ Keyword arguments: pass values according to the name of parameters

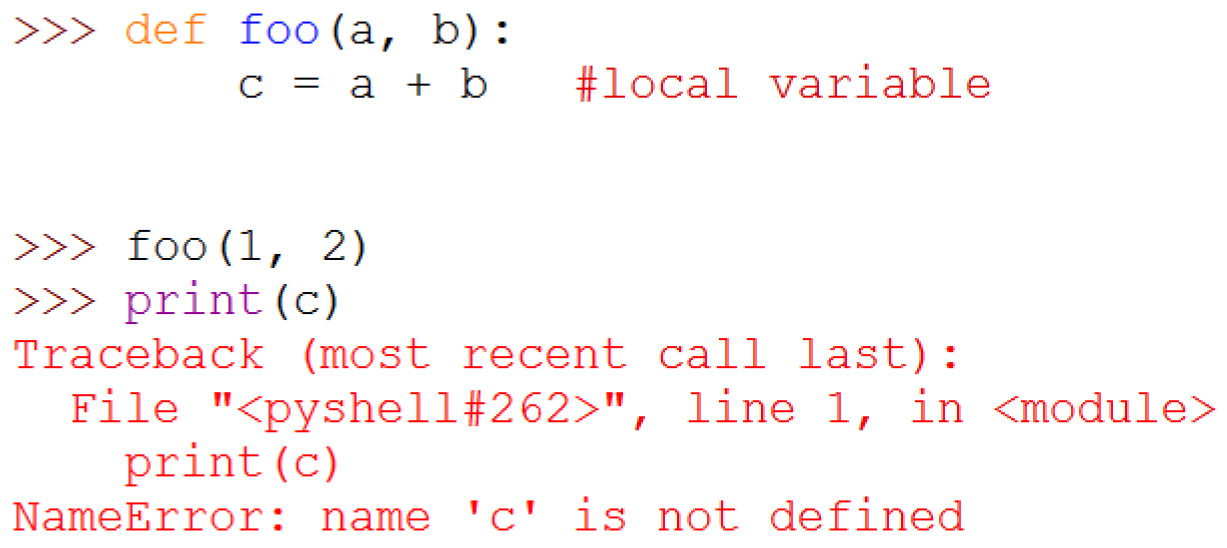
```
funcname([argument_list])
```



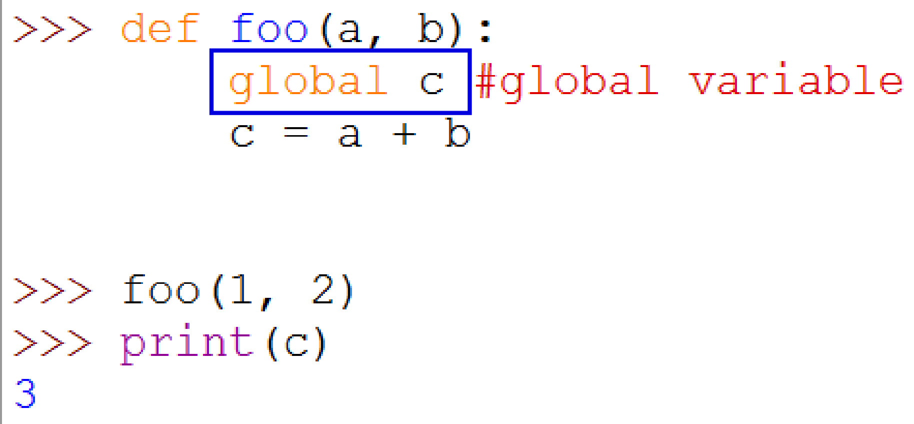
```
>>> def foo(a, b, c):
        print("a = %d" % (a))
        print("b = %d" % (b))
        print("c = %d" % (c))


>>> foo(1, 3, 7)
a = 1
b = 3
c = 7
>>> foo(1, 7, 3)
a = 1
b = 7
c = 3
>>> foo(a=1, c=7, b=3)
a = 1
b = 3
c = 7
>>> foo(1, c=7, b=3)
a = 1
b = 3
c = 7
```

➢ Function

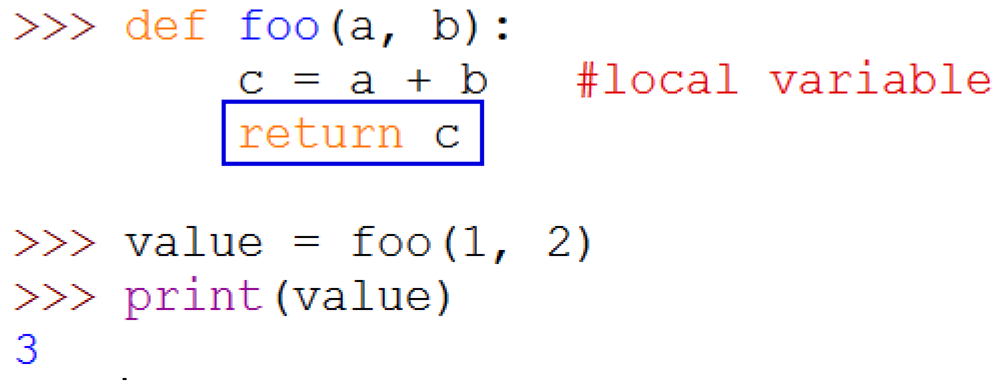
- Variable scope:
  - Local variable: defined inside a function
    - ✓ Only valid in the function
    - ✓ Use return to send back the information in the function
  - Global variable: can be accessed in the entire body of program
    - ✓ It is discouraged, try to avoid



```
>>> def foo(a, b):
        c = a + b    #local variable


>>> foo(1, 2)
>>> print(c)
Traceback (most recent call last):
  File "<pyshell#262>", line 1, in <module>
    print(c)
NameError: name 'c' is not defined
```



```
>>> def foo(a, b):
        global c #global variable
        c = a + b


>>> foo(1, 2)
>>> print(c)
3
```

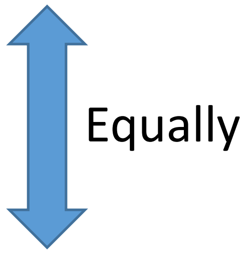


```
>>> def foo(a, b):
        c = a + b    #local variable
        return c

>>> value = foo(1, 2)
>>> print(value)
3
```

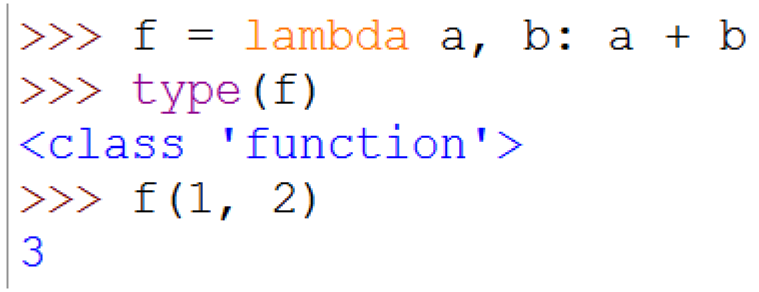
➢ Function

- Lambda expression:
  - Is a disposable, one-line function without a name
  - Used for a short function
  - Actually yields a function, namely anonymous function

```
lambda [parameter_list]: expression
```



```
Def <lambda> ([parameter_list]):
    expression
```



```
>>> f = lambda a, b: a + b
>>> type(f)
<class 'function'>
>>> f(1, 2)
3
```

➢ Input, Output

- Input: pause and read data from the user

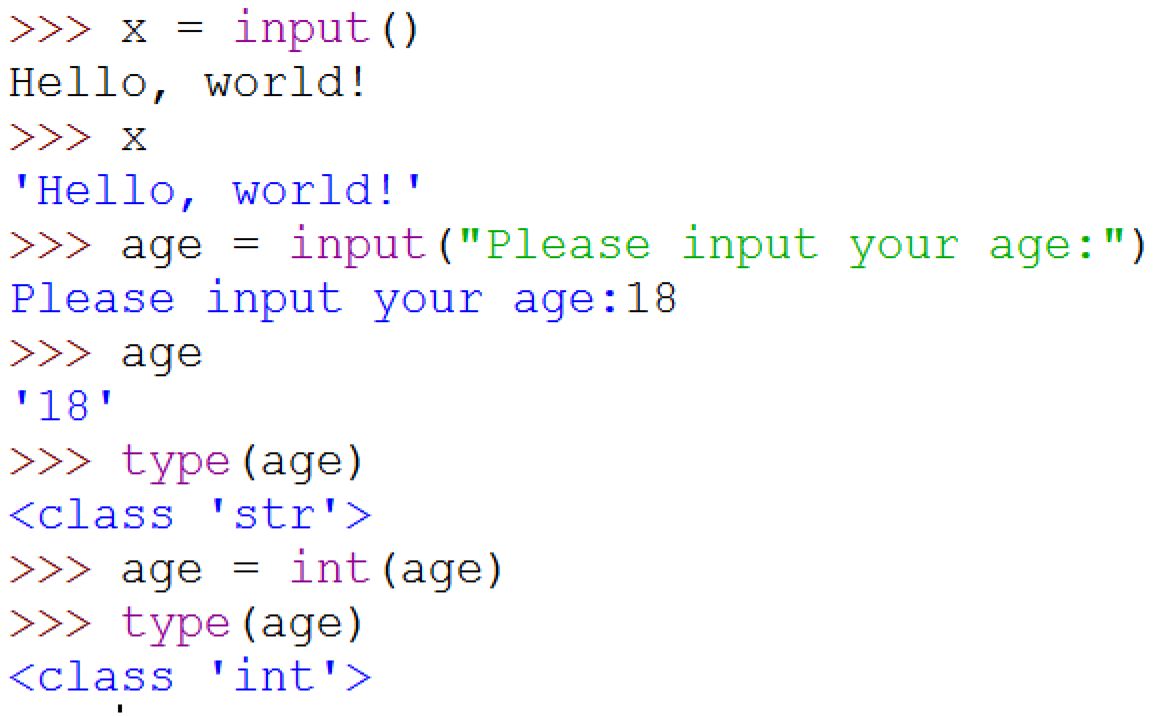
  Usage: input([promt])

  ✓ promt is tips for users

  ✓ promt is not necessary

  ✓ The result of *input* is string type

  ✓ The number can be obtained by explicitly type conversion, such as int(), float()



```
>>> x = input()
Hello, world!
>>> x
'Hello, world!'
>>> age = input("Please input your age:")
Please input your age:18
>>> age
'18'
>>> type(age)
<class 'str'>
>>> age = int(age)
>>> type(age)
<class 'int'>
```

Input, Output

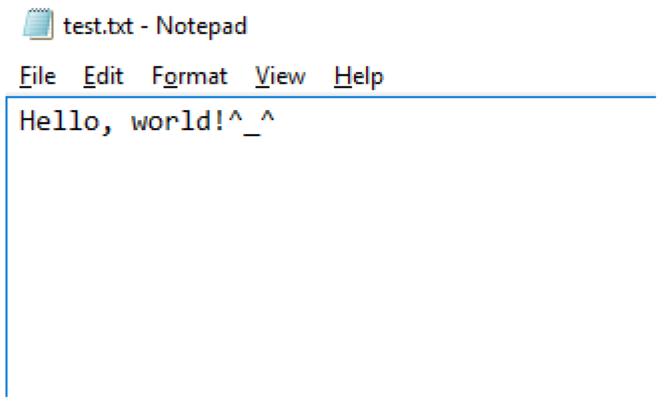
- output: output the information to the given stream

  Usage:

```
print(*values, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- ✓ *values: information to be outputted
- ✓ sep: used to separate the values, default use is space(‘ ’)
- ✓ end: the sign after outputting the information, default use is a new line(‘\n’)
- ✓ file: output steam, can be a file, default is the screen(sys.stdout)
- ✓ sep, end, file, flush, if present, must be given as keyword arguments

```
>>> print(1, 2, 3)
1 2 3
>>> print(1, 2, 3, sep = ',')
1,2,3
>>> print(1, 2, 3, end = '*')
1 2 3*
>>> print("I bought %dkg %s with %f SGD." % (3, 'Orange', 10.5))
I bought 3kg Orange with 10.500000 SGD.
>>> fp = open('D:test.txt', 'w')
>>> fp
<_io.TextIOWrapper name='D:test.txt' mode='w' encoding='cp1252'>
>>> print("Hello, world!", end = '^_^', file = fp, flush=True)
```



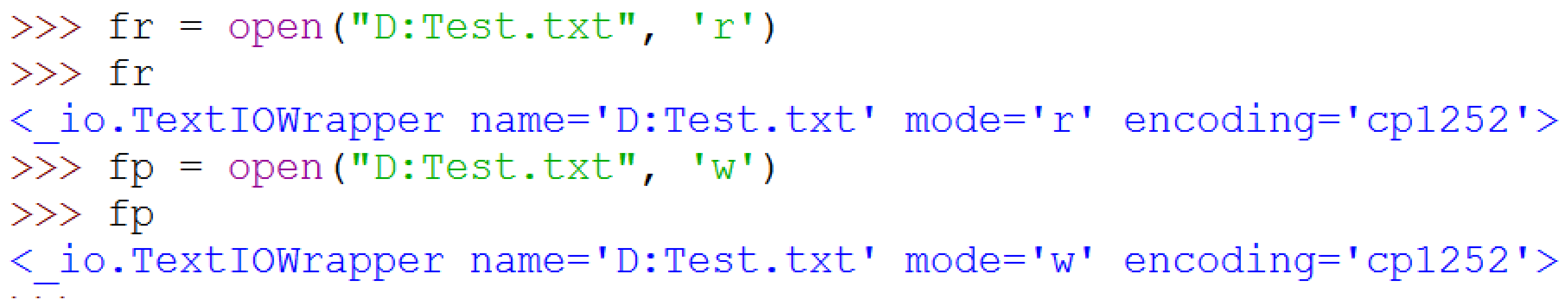
```
Hello, world!^_^
```

File Operations

- File operations include open, read, write, and close
- File open:

  Usage: `handle = open('filename', mode)`

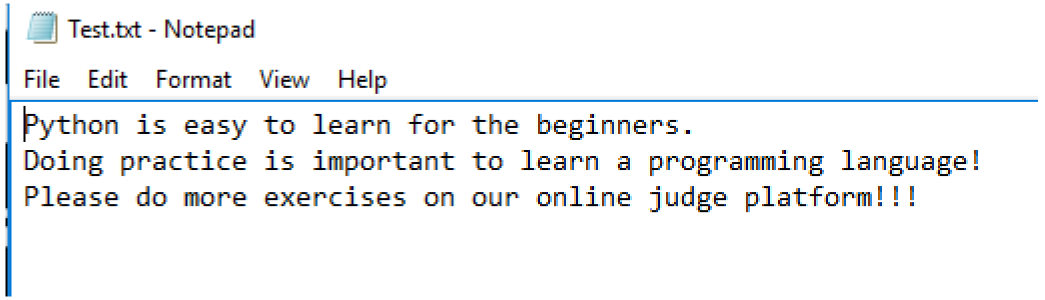
  - mode:
    - ✓ 'r': read-only
    - ✓ 'w': write, remove the old content first
    - ✓ 'a': append the new content after the old one



```
>>> fr = open("D:Test.txt", 'r')
>>> fr
<_io.TextIOWrapper name='D:Test.txt' mode='r' encoding='cp1252'>
>>> fp = open("D:Test.txt", 'w')
>>> fp
<_io.TextIOWrapper name='D:Test.txt' mode='w' encoding='cp1252'>
```

File Operations

- File operations include open, read, write, and close
- File read: read content from the opened file
  - Usage 1: file handle open for read can be treated as a sequence of string
    - ✓ each line in the file is a string in the sequence
    - ✓ Use for statement to iterate through the sequence



```
Python is easy to learn for the beginners.
Doing practice is important to learn a programming language!
Please do more exercises on our online judge platform!!!
```

```
>>> fr = open("D:Test.txt", 'r')
>>> fr
<_io.TextIOWrapper name='D:Test.txt' mode='r' encoding='cp1252'>
>>> for x in fr:
        print(x)


Python is easy to learn for the beginners.

Doing practice is important to learn a programming language!

Please do more exercises on our online judge platform!!!
>>>
```

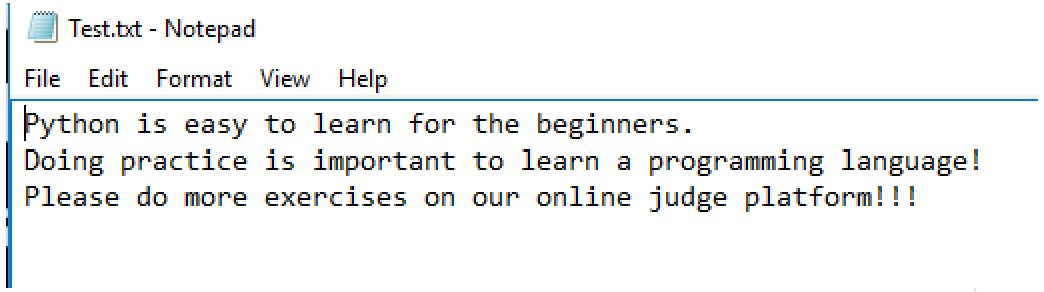
➢ File Operations

- File read: read content from the opened file
  - Usage 2:

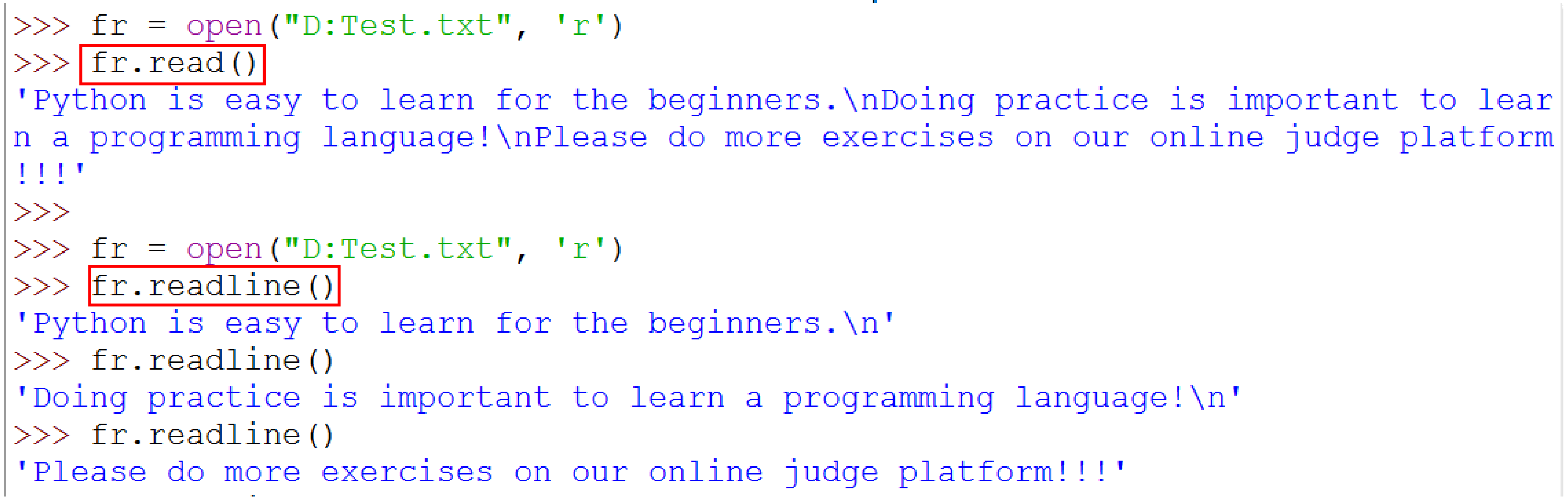
    f.read()
    f.readline()

    ✓ read(): read all content as one string

    ✓ readline(): read one line from the content
  - '\n': special character to indicate the "newline"



```
Python is easy to learn for the beginners.
Doing practice is important to learn a programming language!
Please do more exercises on our online judge platform!!!
```

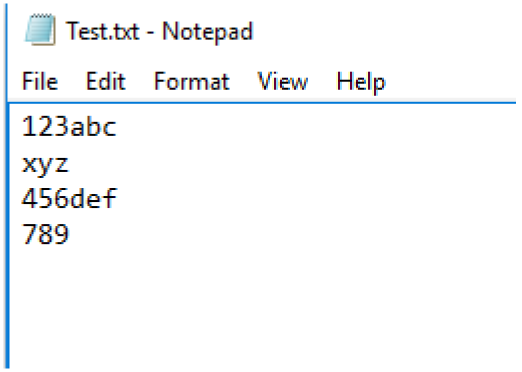


```
>>> fr = open("D:Test.txt", 'r')
>>> fr.read()
'Python is easy to learn for the beginners.\nDoing practice is important to lear
n a programming language!\nPlease do more exercises on our online judge platform
!!!'
>>>
>>> fr = open("D:Test.txt", 'r')
>>> fr.readline()
'Python is easy to learn for the beginners.\n'
>>> fr.readline()
'Doing practice is important to learn a programming language!\n'
>>> fr.readline()
'Please do more exercises on our online judge platform!!!'
```

File Operations

- File write: write the content to the file
  - Usage:
    - ✓ write(s): write the content is s to the file
    - ✓ writelines(lines): write content in lines to the file
    - ✓ flush(): flush the buffer to the file
    - ✓ Note: write()/writelines() does not add the “newline”. Add ‘\n’ explicitly.

```
>>> fw = open("D:\Test.txt", 'w')
>>> fw
<_io.TextIOWrapper name='D:\\Test.txt' mode='w' encoding='cp1252'>
>>> fw.write("123")
3
>>> fw.write('abc\n')
4
>>> fw.write('xyz\n')
4
>>> fw.writelines(['456', 'def\n', '789'])
>>> fw.flush()
>>> fw.close()
```

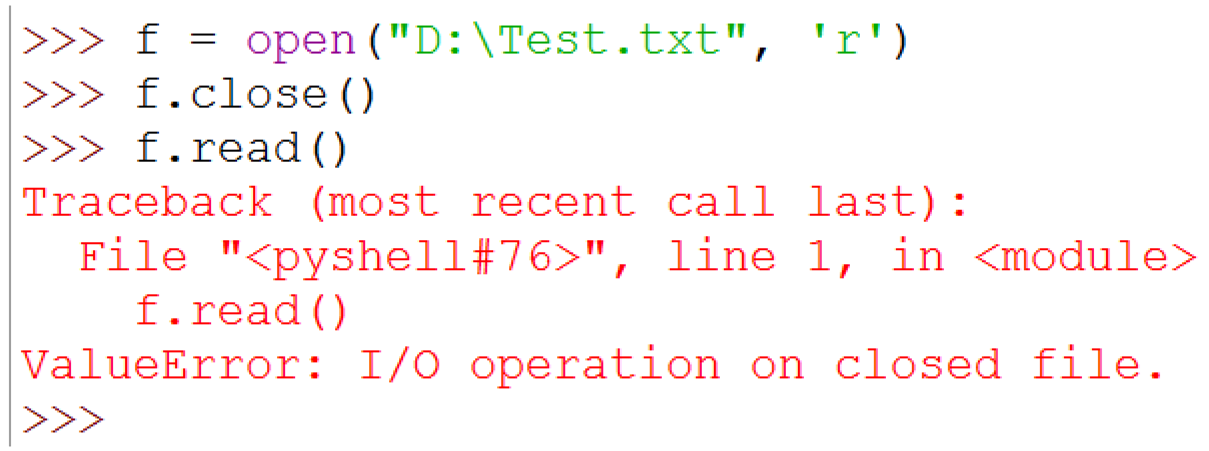


```
123abc
xyz
456def
789
```

➢ File Operations

- File operations include open, read, write, and close
- File close: close the file after finishing the file operations to free up the resource

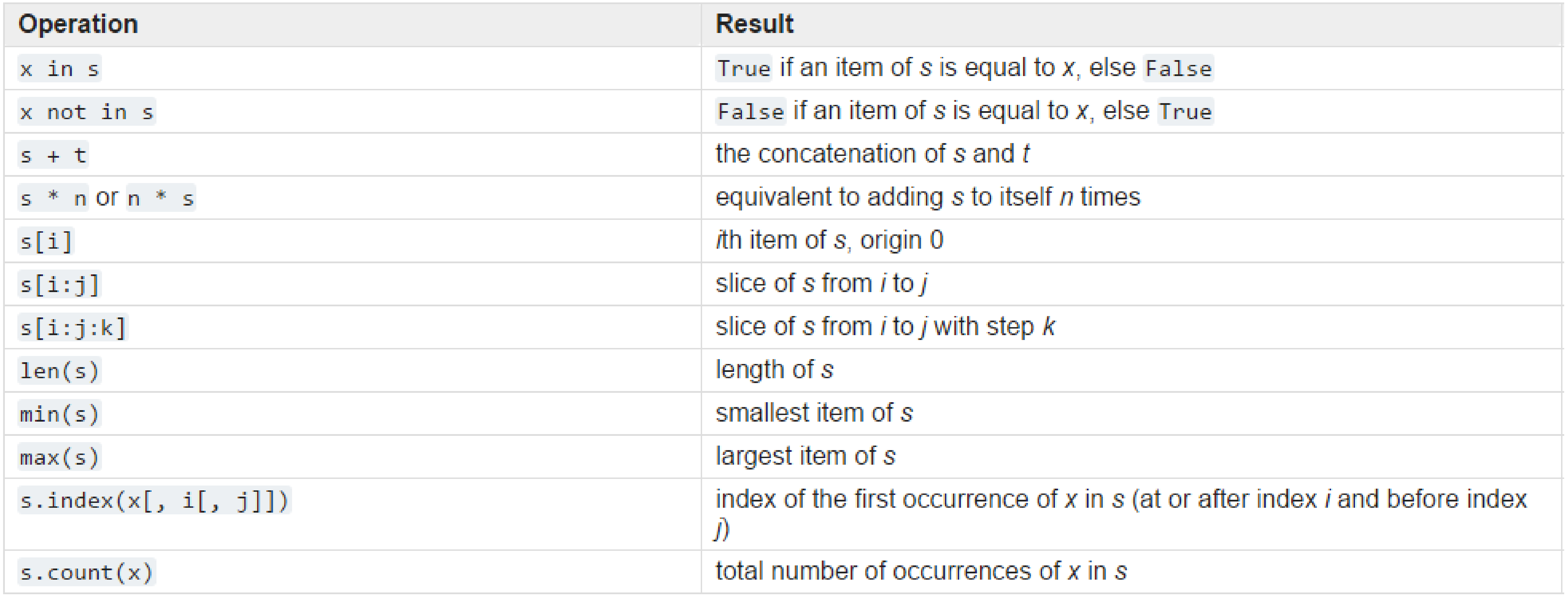
  Usage: f.close()



```
>>> f = open("D:\Test.txt", 'r')
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<pyshell#76>", line 1, in <module>
    f.read()
ValueError: I/O operation on closed file.
>>>
```

➢ Data Structure---Sequence

- Sequence is a type of collection, which allows to put many values in a single ”variable”.
- Common sequence operations:



Operation	Result
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)
`s.count(x)`	total number of occurrences of *x* in *s*

➢ Data Structure---Sequence

- Useful sequence types are: list, tuple, string
- The element in a sequence can be visited via index: *s*[*i*]
  - The index begins from 0, must be integer
  - The index can not exceed the length of sequence

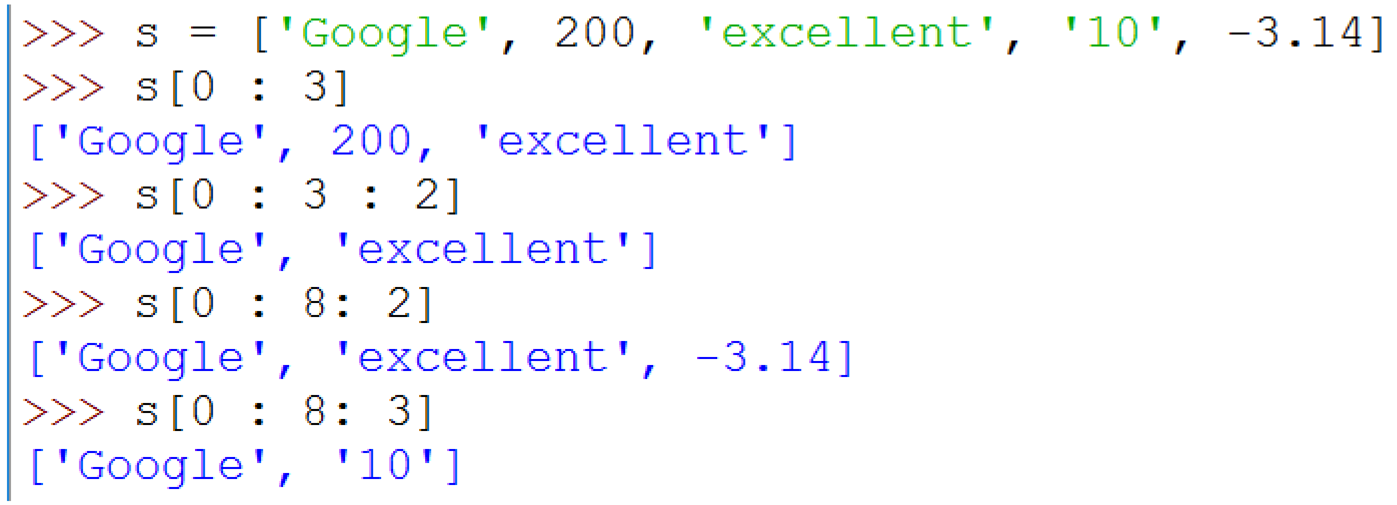
‘Google’	200	‘excellent’	‘10’	-3.14
s[0]	s[1]	s[2]	s[3]	s[4]

```
>>> s = ['Google', 200, 'excellent', '10', -3.14]
>>> len(s)
5
>>> 'Google' in s
True
>>> 'Apple' in s
False
>>> s[0]
'Google'
>>> s[3]
'10'
>>> s[6]
Traceback (most recent call last):
  File "<pyshell#90>", line 1, in <module>
    s[6]
IndexError: list index out of range
>>> s[1.5]
Traceback (most recent call last):
  File "<pyshell#91>", line 1, in <module>
    s[1.5]
TypeError: list indices must be integers, not float
```

➢ Data Structure---Sequence

- Slice: grab sections of a sequence, s[i : j] or s[i: j : k]
  - i: the start of the index
  - j: the end of index, but never be included. j can be larger than the length of sequence
  - k: the step size

‘Google’	200	‘excellent’	‘10’	-3.14
s[0]	s[1]	s[2]	s[3]	s[4]

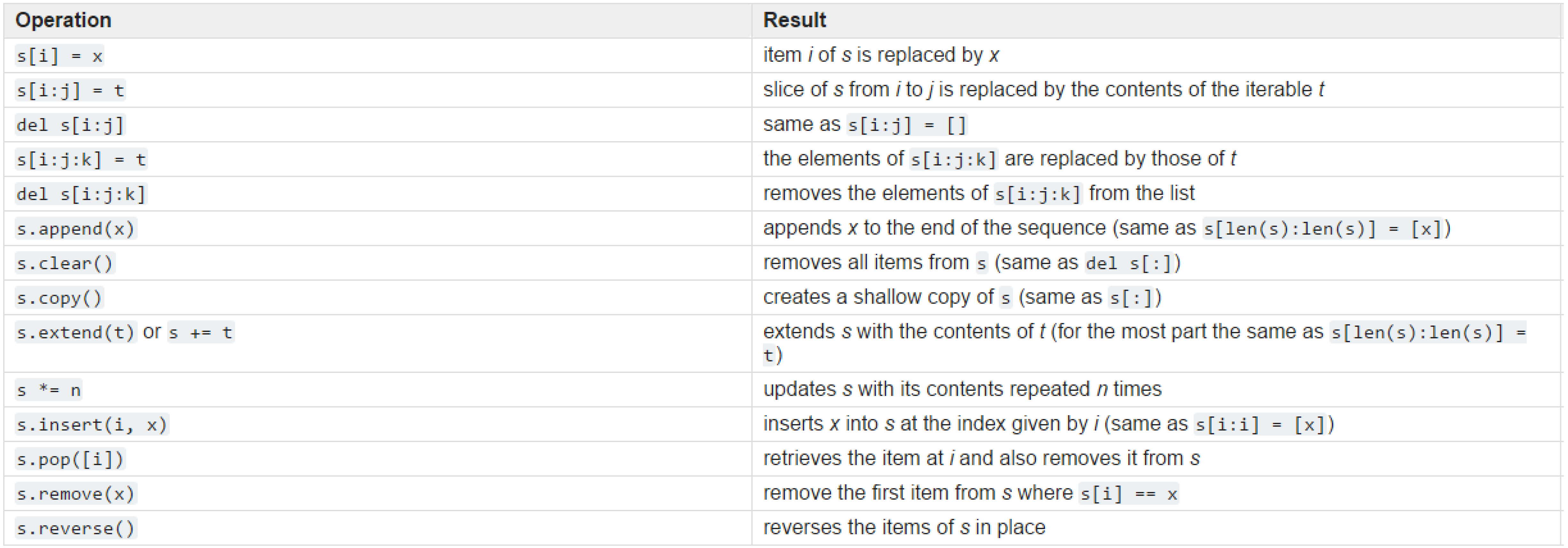


```
>>> s = ['Google', 200, 'excellent', '10', -3.14]
>>> s[0 : 3]
['Google', 200, 'excellent']
>>> s[0 : 3 : 2]
['Google', 'excellent']
>>> s[0 : 8: 2]
['Google', 'excellent', -3.14]
>>> s[0 : 8: 3]
['Google', '10']
```

➢ Data Structure---Sequence: List

- List: [v1, v2, …, vn]
  - List is mutable, namely the values of elements can be changed
  - List can be sorted

The operations defined on mutable sequence types



Operation	Result
`s[i] = x`	item *i* of *s* is replaced by *x*
`s[i:j] = t`	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*
`del s[i:j]`	same as `s[i:j] = []`
`s[i:j:k] = t`	the elements of `s[i:j:k]` are replaced by those of *t*
`del s[i:j:k]`	removes the elements of `s[i:j:k]` from the list
`s.append(x)`	appends *x* to the end of the sequence (same as `s[len(s):len(s)] = [x]`)
`s.clear()`	removes all items from `s` (same as `del s[:]`)
`s.copy()`	creates a shallow copy of `s` (same as `s[:]`)
`s.extend(t)` or `s += t`	extends *s* with the contents of *t* (for the most part the same as `s[len(s):len(s)] = t`)
`s *= n`	updates *s* with its contents repeated *n* times
`s.insert(i, x)`	inserts *x* into *s* at the index given by *i* (same as `s[i:i] = [x]`)
`s.pop([i])`	retrieves the item at *i* and also removes it from *s*
`s.remove(x)`	remove the first item from *s* where `s[i] == x`
`s.reverse()`	reverses the items of *s* in place

➢ Data Structure---Sequence: List

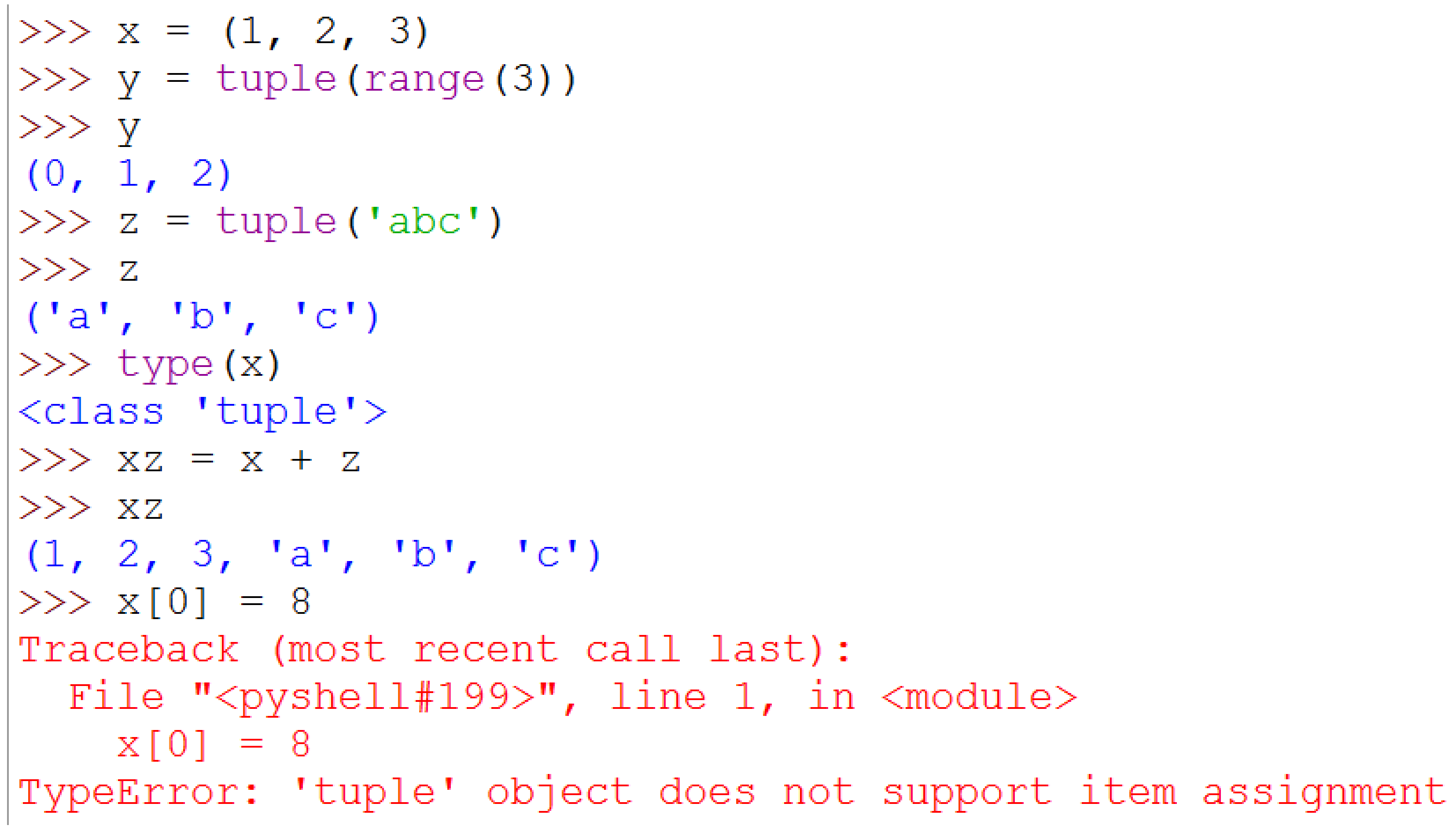
- List: [v1, v2, …, vn]
  - List is mutable, namely the values of elements can be changed
  - List can be sorted

```
>>> s = [] #empty list
>>> s
[]
>>> s = [1, 2, 3] #another way
>>> s
[1, 2, 3]
>>> s = [x for x in range(4)] #another way
>>> s
[0, 1, 2, 3]
>>> len(s)
4
>>> max(s)
3
>>> sum(s)
6
>>> sum(s) / len(s) #get average
1.5
```

```
>>> s = [1, 2, 3] #another way
>>> s
[1, 2, 3]
>>> s *= 2
>>> s
[1, 2, 3, 1, 2, 3]
>>> s[1] = 10 #mutable
>>> s
[1, 10, 3, 1, 2, 3]
>>> del s[2 : 4] #remove elements
>>> s
[1, 10, 2, 3]
>>> s.append(5) #append 5 at the end
>>> s
[1, 10, 2, 3, 5]
>>> s.insert(1, 8) #insert 8 at index 1
>>> s
[1, 8, 10, 2, 3, 5]
>>> s.reverse()
>>> s
[5, 3, 2, 10, 8, 1]
>>> s.sort()
>>> s
[1, 2, 3, 5, 8, 10]
```

➢ Data Structure---Sequence: Tuple

- Tuple: (v1, v2, …, vn)
  - Tuple is immutable, namely the elements can not be changed

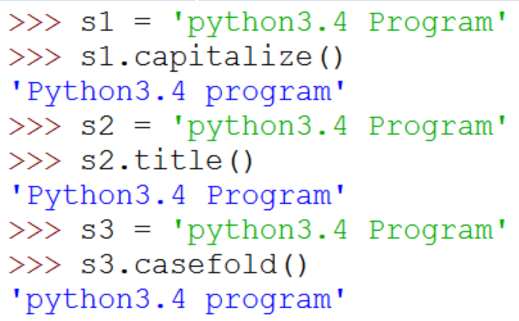


```
>>> x = (1, 2, 3)
>>> y = tuple(range(3))
>>> y
(0, 1, 2)
>>> z = tuple('abc')
>>> z
('a', 'b', 'c')
>>> type(x)
<class 'tuple'>
>>> xz = x + z
>>> xz
(1, 2, 3, 'a', 'b', 'c')
>>> x[0] = 8
Traceback (most recent call last):
  File "<pyshell#199>", line 1, in <module>
    x[0] = 8
TypeError: 'tuple' object does not support item assignment
```

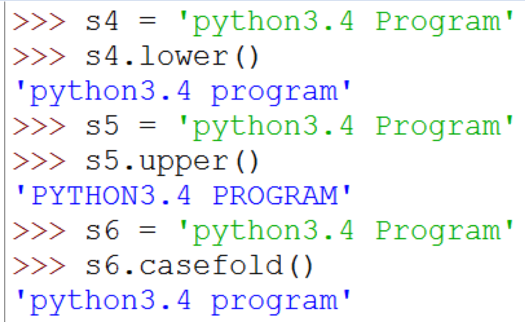
➤ Data Structure---Sequence: String

- String: ‘…’, “…”
  - String is immutable, namely the elements can not be changed
  - String is a sequence, so the common operations of sequence apply to string
  - String has many additional operations:

Change between lowercase and uppercase	
Operation	Result
capitalize()	Return a copy of the string with its first character capitalized and the rest lowercased.
lower()	Return a copy of the string with all the cased characters converted to lowercase.
upper()	Return a copy of the string with all the cased characters converted to uppercase.
swapcase()	Return a copy of the string with uppercase characters converted to lowercase and vice versa.
title()	Return a titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase.
casefold()	Return a casefolded copy of the string. Casefolded strings may be used for caseless matching.



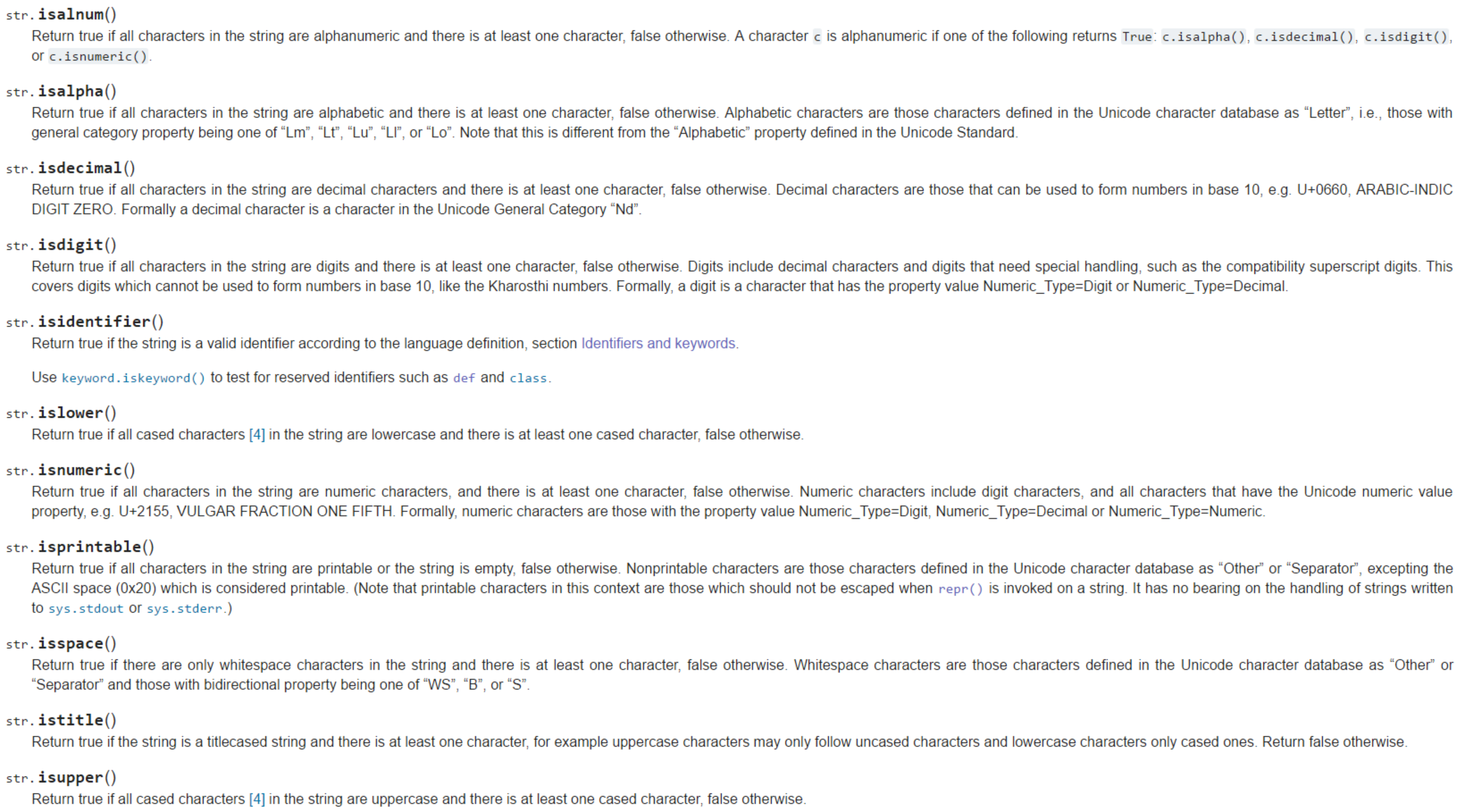
```
>>> s1 = 'python3.4 Program'
>>> s1.capitalize()
'Python3.4 program'
>>> s2 = 'python3.4 Program'
>>> s2.title()
'Python3.4 Program'
>>> s3 = 'python3.4 Program'
>>> s3.casefold()
'python3.4 program'
```



```
>>> s4 = 'python3.4 Program'
>>> s4.lower()
'python3.4 program'
>>> s5 = 'python3.4 Program'
>>> s5.upper()
'PYTHON3.4 PROGRAM'
>>> s6 = 'python3.4 Program'
>>> s6.casefold()
'python3.4 program'
```

➢ Data Structure---Sequence: String

- String has many additional operations: Type Check



`str.`**`isalnum`**`()`

Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise. A character `c` is alphanumeric if one of the following returns `True`: `c.isalpha()`, `c.isdecimal()`, `c.isdigit()`, or `c.isnumeric()`.

`str.`**`isalpha`**`()`

Return true if all characters in the string are alphabetic and there is at least one character, false otherwise. Alphabetic characters are those characters defined in the Unicode character database as "Letter", i.e., those with general category property being one of "Lm", "Lt", "Lu", "Ll", or "Lo". Note that this is different from the "Alphabetic" property defined in the Unicode Standard.

`str.`**`isdecimal`**`()`

Return true if all characters in the string are decimal characters and there is at least one character, false otherwise. Decimal characters are those that can be used to form numbers in base 10, e.g. U+0660, ARABIC-INDIC DIGIT ZERO. Formally a decimal character is a character in the Unicode General Category "Nd".

`str.`**`isdigit`**`()`

Return true if all characters in the string are digits and there is at least one character, false otherwise. Digits include decimal characters and digits that need special handling, such as the compatibility superscript digits. This covers digits which cannot be used to form numbers in base 10, like the Kharosthi numbers. Formally, a digit is a character that has the property value Numeric_Type=Digit or Numeric_Type=Decimal.

`str.`**`isidentifier`**`()`

Return true if the string is a valid identifier according to the language definition, section Identifiers and keywords.

Use `keyword.iskeyword()` to test for reserved identifiers such as `def` and `class`.

`str.`**`islower`**`()`

Return true if all cased characters [4] in the string are lowercase and there is at least one cased character, false otherwise.

`str.`**`isnumeric`**`()`

Return true if all characters in the string are numeric characters, and there is at least one character, false otherwise. Numeric characters include digit characters, and all characters that have the Unicode numeric value property, e.g. U+2155, VULGAR FRACTION ONE FIFTH. Formally, numeric characters are those with the property value Numeric_Type=Digit, Numeric_Type=Decimal or Numeric_Type=Numeric.

`str.`**`isprintable`**`()`

Return true if all characters in the string are printable or the string is empty, false otherwise. Nonprintable characters are those characters defined in the Unicode character database as "Other" or "Separator", excepting the ASCII space (0x20) which is considered printable. (Note that printable characters in this context are those which should not be escaped when `repr()` is invoked on a string. It has no bearing on the handling of strings written to `sys.stdout` or `sys.stderr`.)

`str.`**`isspace`**`()`

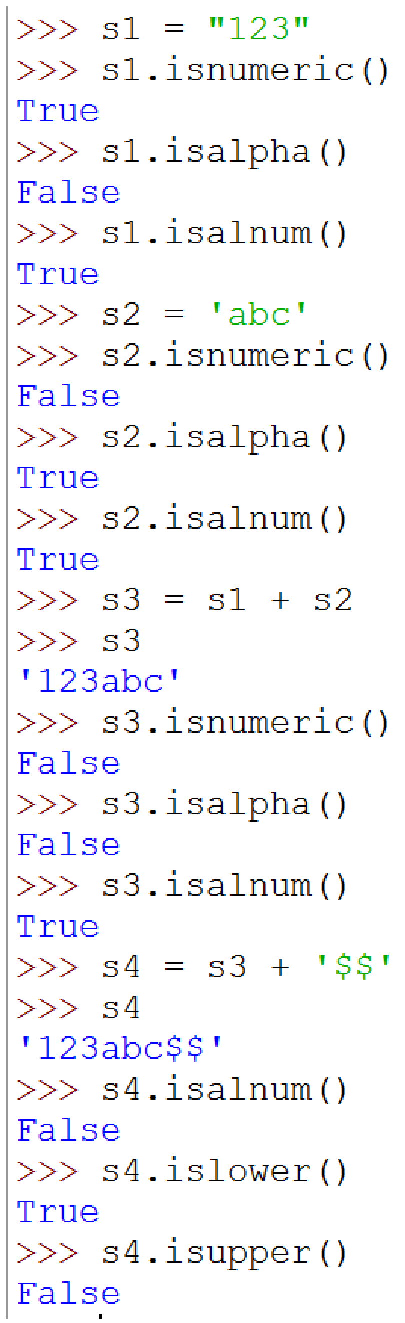
Return true if there are only whitespace characters in the string and there is at least one character, false otherwise. Whitespace characters are those characters defined in the Unicode character database as "Other" or "Separator" and those with bidirectional property being one of "WS", "B", or "S".

`str.`**`istitle`**`()`

Return true if the string is a titlecased string and there is at least one character, for example uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return false otherwise.

`str.`**`isupper`**`()`

Return true if all cased characters [4] in the string are uppercase and there is at least one cased character, false otherwise.



```
>>> s1 = "123"
>>> s1.isnumeric()
True
>>> s1.isalpha()
False
>>> s1.isalnum()
True
>>> s2 = 'abc'
>>> s2.isnumeric()
False
>>> s2.isalpha()
True
>>> s2.isalnum()
True
>>> s3 = s1 + s2
>>> s3
'123abc'
>>> s3.isnumeric()
False
>>> s3.isalpha()
False
>>> s3.isalnum()
True
>>> s4 = s3 + '$$'
>>> s4
'123abc$$'
>>> s4.isalnum()
False
>>> s4.islower()
True
>>> s4.isupper()
False
```

➢ Data Structure---Sequence: String

Operation	Result
strip([*chars*])	Return a copy of the string with the leading and trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed.
lstrip([*chars*])	Return a copy of the string with leading characters removed.
count(*sub*[, *start*[, *end*]])	Return the number of non-overlapping occurrences of substring *sub* in the range [start, end]. Optional arguments *start* and *end* are interpreted as in slice notation.
startswith(*prefix*[, *start*[, *end*]])	Return True if string starts with the *prefix*, otherwise return False.
endswith(*suffix*[, *start*[, *end*]])	Return True if the string ends with the specified *suffix*, otherwise return False.
find(*sub*[, *start*[, *end*]])	Return the lowest index in the string where substring *sub* is found within the slice s[start : end]. -1 is returned if *sub* is not found.
replace(*old*, *new*[, *count*])	Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

```
>>> s1 = '  Googlegogogo  '
>>> s1
'  Googlegogogo  '
>>> s2 = s1.strip(' ')
>>> s2
'Googlegogogo'
>>> s3 = s1.lstrip(' ')
>>> s3
'Googlegogogo  '
```

```
>>> s2
'Googlegogogo'
>>> s2.count('g')
4
>>> s2[0 : 7]
'Googleg'
>>> s2.count('g', 0, 7)
2
>>> s2.startswith('Goo')
True
>>> s2.endswith('goo')
False
```

```
>>> s2
'Googlegogogo'
>>> s2.find('go')
6
>>> s2[0 : 6]
'Google'
>>> s2[6 : len(s2)]
'gogogo'
>>> s2.find('goo')
-1
>>> s2.replace('go', 'AbC')
'GoogleAbCAbCAbC'
```

➢ Data Structure---Sequence: String

Operation	Result
split(*sep=None*, *maxsplit=-1*)	Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done (thus, the list will have at most *maxsplit*+1 elements). If *maxsplit* is not specified or -1, then there is no limit on the number of splits (all possible splits are made).
rsplit(*sep=None*, *maxsplit=-1*)	Similar to split(), but starts from the rightmost side.
partition(*sep*)	Split the string at the first occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator.
join(*iterable*)	Return a string which is the concatenation of the strings in the iterable *iterable*.

```
>>> s = 'one,two,three'
>>> s.split(',')
['one', 'two', 'three']
>>> s.split(',', 1)
['one', 'two,three']
>>> s.rsplit(',', 1)
['one,two', 'three']
>>> s.partition(',')
('one', ',', 'two,three')
>>> s2 = ('a', 'b', 'c')
>>> s3 = ':'
>>> s3.join(s2)
'a:b:c'
>>> s3.join('Python')
'P:y:t:h:o:n'
```

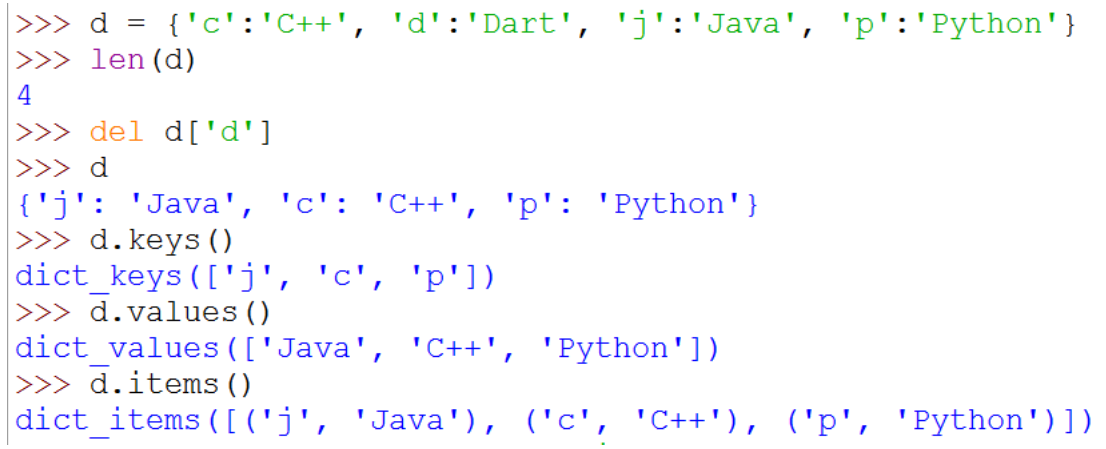
➢ Data Structure---Dictionary

- Dictionary: maintain a mapping relationship between key and value
  - {key1 : value1, …, keyn : valuen}
  - Keys must be unique
  - Values are visited, modified, added using [key]
  - [key], Error to visit a key which does not exist; should use in to check the existence first

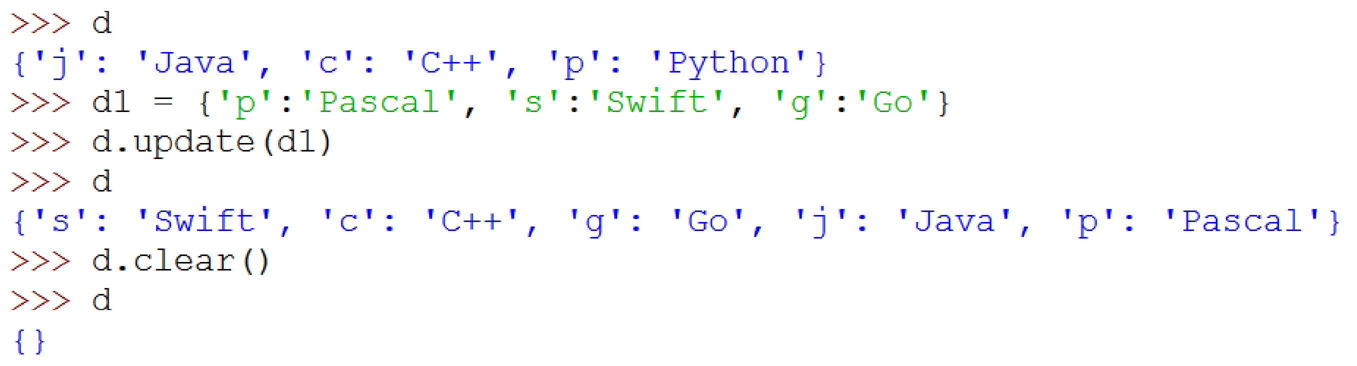
```
>>> x = {'c':'C++', 'd':'Dart', 'j':'Java', 'p':'Python'}
>>> type(x)
<class 'dict'>
>>> x['p'] #access
'Python'
>>> x['p'] = 'Pascal' #modify
>>> x['p']
'Pascal'
>>> x
{'j': 'Java', 'd': 'Dart', 'c': 'C++', 'p': 'Pascal'}
>>> x['s'] = 'Swift' #add new pair
>>> x
{'s': 'Swift', 'j': 'Java', 'd': 'Dart', 'c': 'C++', 'p': 'Pascal'}
>>> x['b'] #visit a key which does not exist, Error
Traceback (most recent call last):
  File "<pyshell#399>", line 1, in <module>
    x['b'] #visit a key which does not exist, Error
KeyError: 'b'
>>> 'b' in x
False
```

➢ Data Structure---Dictionary

Operation	Result
len(d)	Return the number of items in the dictionary *d*.
del d[key]	Remove d[key] from *d*.
key in d	Return True if d has a key *key*, else False.
clear()	Remove all items from the dictionary.
keys()	Return a new view of the dictionary’s keys.
values()	Return a new view of the dictionary’s values.
items()	Return a new view of the dictionary’s items ((key, value) pairs).
get(*key*[, *default*])	Return the value for key if key is in the dictionary, else default.
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys. Return None.



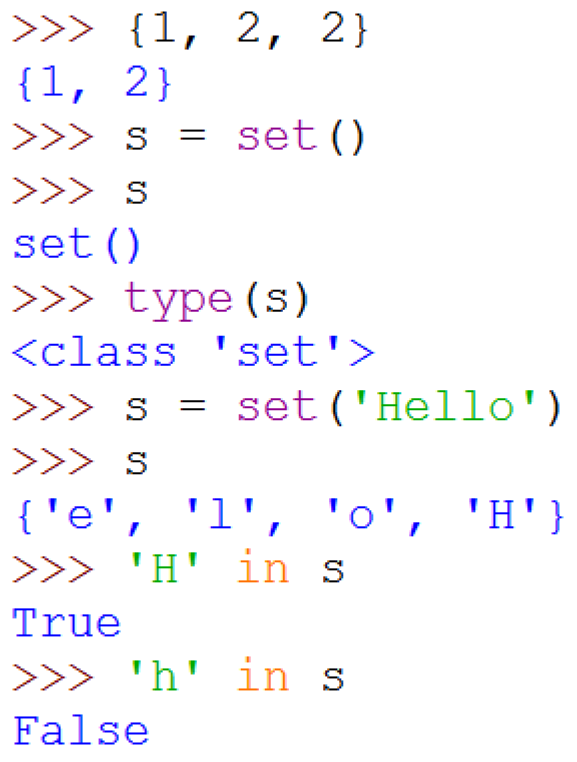
```
>>> d = {'c':'C++', 'd':'Dart', 'j':'Java', 'p':'Python'}
>>> len(d)
4
>>> del d['d']
>>> d
{'j': 'Java', 'c': 'C++', 'p': 'Python'}
>>> d.keys()
dict_keys(['j', 'c', 'p'])
>>> d.values()
dict_values(['Java', 'C++', 'Python'])
>>> d.items()
dict_items([('j', 'Java'), ('c', 'C++'), ('p', 'Python')])
```



```
>>> d
{'j': 'Java', 'c': 'C++', 'p': 'Python'}
>>> d1 = {'p':'Pascal', 's':'Swift', 'g':'Go'}
>>> d.update(d1)
>>> d
{'s': 'Swift', 'c': 'C++', 'g': 'Go', 'j': 'Java', 'p': 'Pascal'}
>>> d.clear()
>>> d
{}
```

Data Structure---Set

- Set: an unordered collection of distinct hashable objects
  - {x1, …, xn}
  - Values in a set must be unique
  - Operations in, not in are used to check whether an element is in the set



```
>>> {1, 2, 2}
{1, 2}
>>> s = set()
>>> s
set()
>>> type(s)
<class 'set'>
>>> s = set('Hello')
>>> s
{'e', 'l', 'o', 'H'}
>>> 'H' in s
True
>>> 'h' in s
False
```

Data Structure---Set

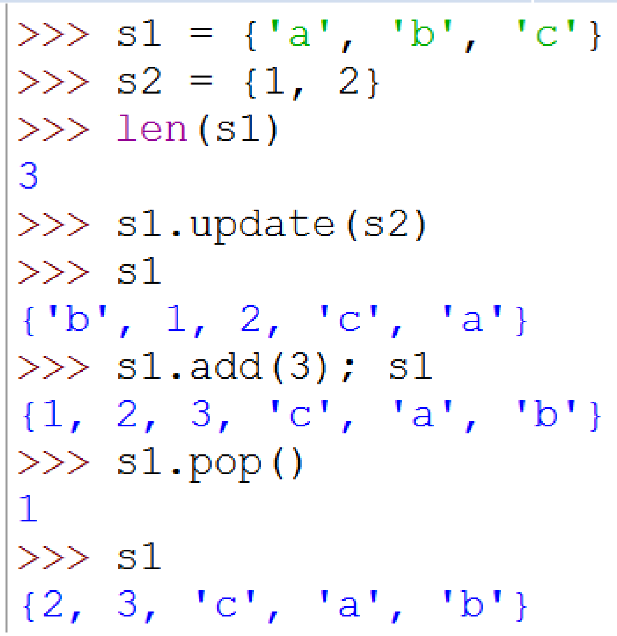
Operator	Operation	Result
s1 | s2 |…	union(*others)	Return a new set with elements from the set and all others ： s1 ∪ s2 ∪…
s1 & s2 & …	intersection(*others)	Return a new set with elements common to the set and all others ： s1 ∩ s2 ∩…
s1 – s2 – ….	difference(**others*)	Return a new set with elements in the set that are not in the others ： s1 \ s2 \…
s1 ^ s2	symmetric_difference(other)	Return a new set with elements in either the set or other but not both. ： s1△s2
	isdisjoint(other)	Return True if the set has no elements in common with other.
set <= other	issubset(other)	Test whether every element in the set is in *other*.
set < other		Test whether the set is a proper subset of *other.*
set >= other	issuperset(*other*)	Test whether every element in *other* is in the *set.*
set > other		Test whether the set is a proper superset of *other.*

```
>>> s1 = {1, 2, 3}
>>> s2 = {2, 3, 4}
>>> s1 | s2
{1, 2, 3, 4}
>>> s1 & s2
{2, 3}
>>> s1 - s2
{1}
>>> s1 ^ s2
{1, 4}
```

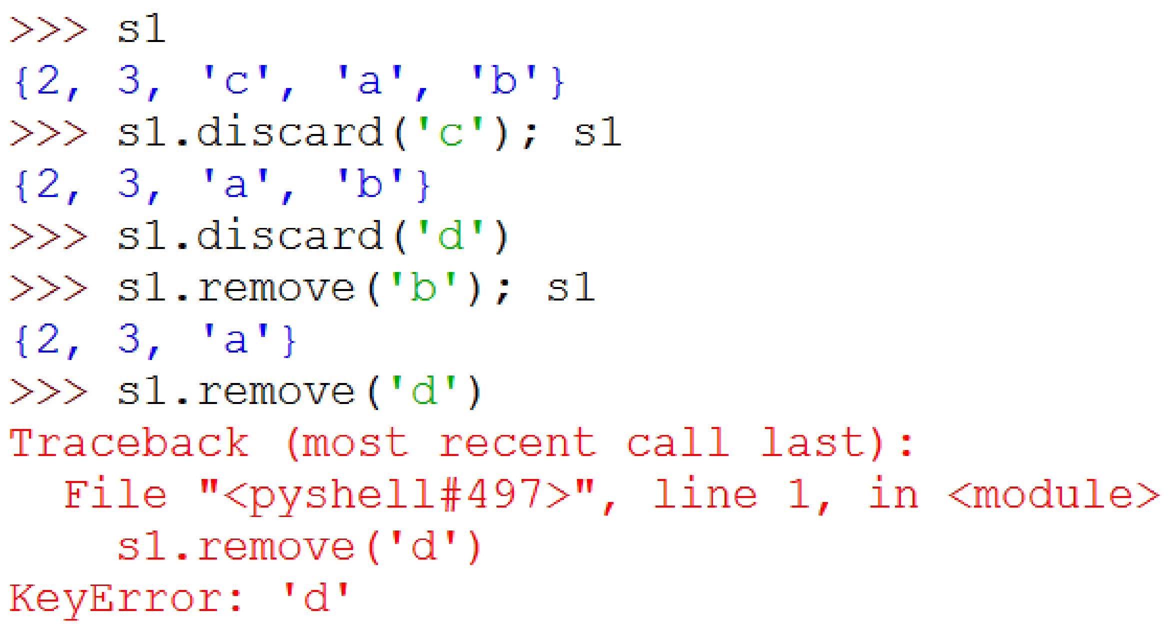
```
>>> s3 = {1, 2, 3}
>>> s4 = {2, 3}
>>> s5 = {3, 4, 5}
>>> s3.isdisjoint(s5)
False
>>> s4 < s3
True
>>> s3.issuperset(s4)
True
```

➢ Data Structure---Set

Operation	Result
len(s)	Return the number of elements in set s (cardinality of s).
update(*others)	Update the set, adding elements from all others.
add(elem)	Add element elem to the set.
remove(elem)	Remove element elem from the set.
discard(*elem*)	Remove element elem from the set if it is present.
pop()	Remove and return an arbitrary element from the set.
clear()	Remove all elements from the set.



```
>>> s1 = {'a', 'b', 'c'}
>>> s2 = {1, 2}
>>> len(s1)
3
>>> s1.update(s2)
>>> s1
{'b', 1, 2, 'c', 'a'}
>>> s1.add(3); s1
{1, 2, 3, 'c', 'a', 'b'}
>>> s1.pop()
1
>>> s1
{2, 3, 'c', 'a', 'b'}
```

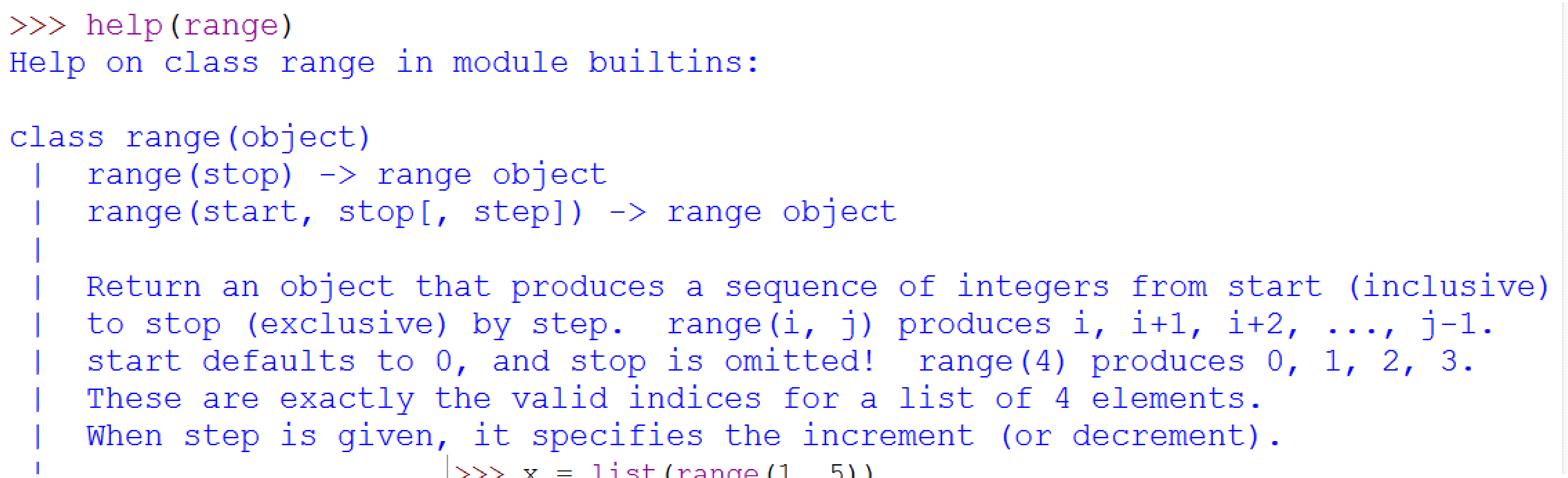


```
>>> s1
{2, 3, 'c', 'a', 'b'}
>>> s1.discard('c'); s1
{2, 3, 'a', 'b'}
>>> s1.discard('d')
>>> s1.remove('b'); s1
{2, 3, 'a'}
>>> s1.remove('d')
Traceback (most recent call last):
  File "<pyshell#497>", line 1, in <module>
    s1.remove('d')
KeyError: 'd'
```

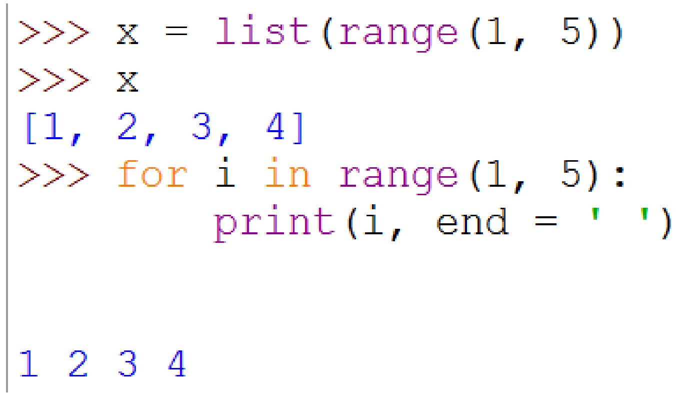
➢ Iterator and Generator---range

- Range(i, j):

  produces the numbers i, i+1, …, j-1.



```
>>> help(range)
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start (inclusive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2, ..., j-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0, 1, 2, 3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |
```

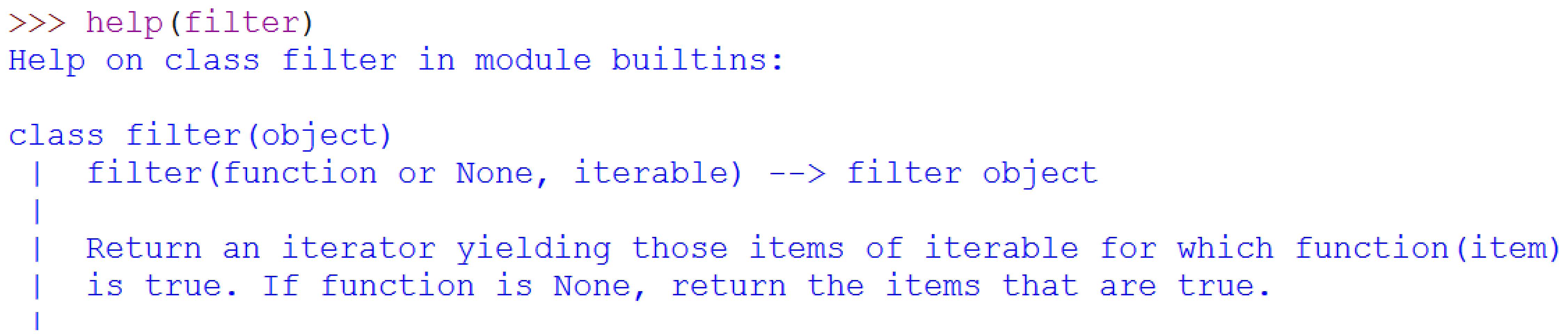


```
>>> x = list(range(1, 5))
>>> x
[1, 2, 3, 4]
>>> for i in range(1, 5):
        print(i, end = ' ')


1 2 3 4
```

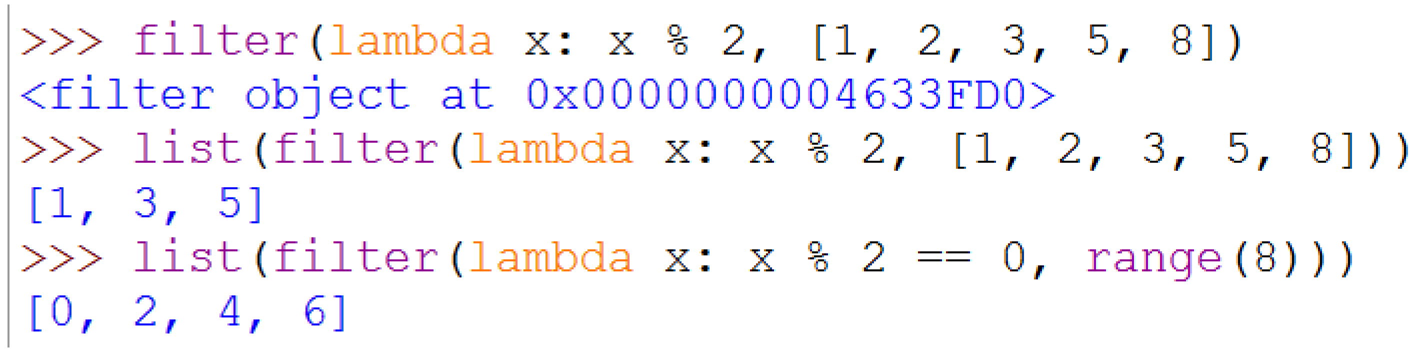
➢ Iterator and Generator---filter

- filter(function, list):

offers an elegant way to filter out all the elements of a list, for which the function *function* returns True.



```
>>> help(filter)
Help on class filter in module builtins:

class filter(object)
 |  filter(function or None, iterable) --> filter object
 |
 |  Return an iterator yielding those items of iterable for which function(item)
 |  is true. If function is None, return the items that are true.
 |
```



```
>>> filter(lambda x: x % 2, [1, 2, 3, 5, 8])
<filter object at 0x0000000004633FD0>
>>> list(filter(lambda x: x % 2, [1, 2, 3, 5, 8]))
[1, 3, 5]
>>> list(filter(lambda x: x % 2 == 0, range(8)))
[0, 2, 4, 6]
```

➢ Iterator and Generator---zip

- zip(**iterables*):

returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables. The iterator stops when the shortest input iterable is exhausted.

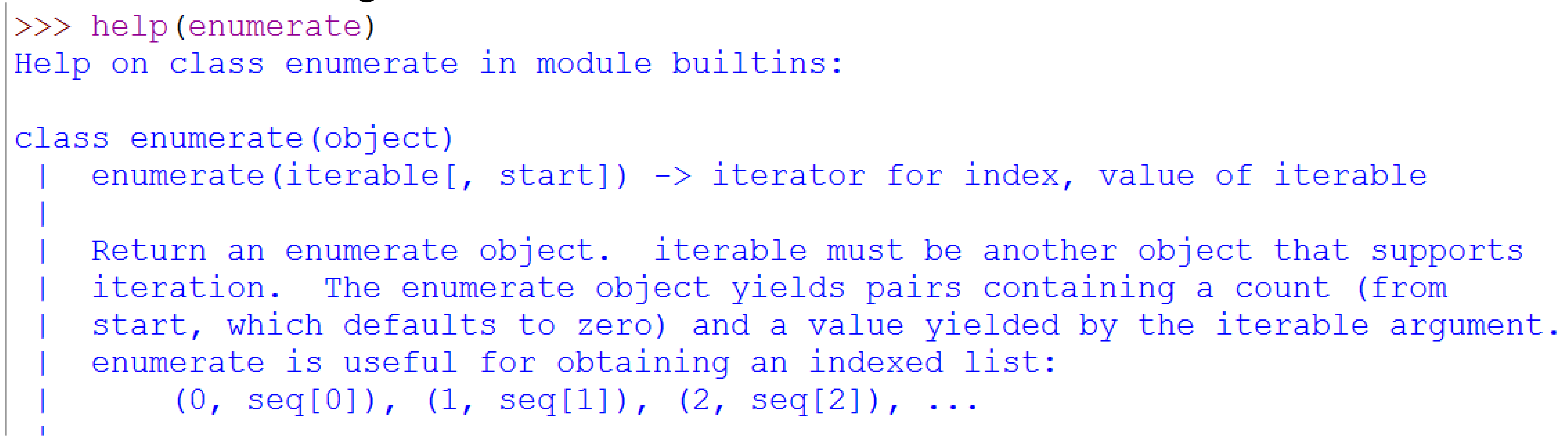
```
>>> help(zip)
Help on class zip in module builtins:

class zip(object)
 |  zip(iter1 [,iter2 [...]]) --> zip object
 |
 |  Return a zip object whose .__next__() method returns a tuple where
 |  the i-th element comes from the i-th iterable argument.  The .__next__()
 |  method continues until the shortest iterable in the argument sequence
 |  is exhausted and then it raises StopIteration.
```

```
>>> zip((1, 2, 3), "ab", range(3))
<zip object at 0x00000000043AB748>
>>> list(zip((1, 2, 3), "ab", range(3)))
[(1, 'a', 0), (2, 'b', 1)]
```

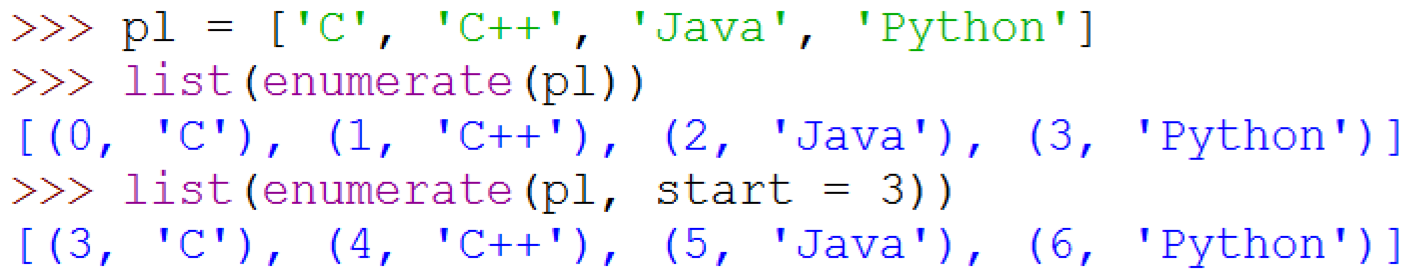
➢ Iterator and Generator---enumerate

- enumerate(*iterable*, *start*=0):

returns a tuple containing a count (from *start* which defaults to 0) and the values obtained from iterating over *iterable*.



```
>>> help(enumerate)
Help on class enumerate in module builtins:

class enumerate(object)
 |  enumerate(iterable[, start]) -> iterator for index, value of iterable
 |
 |  Return an enumerate object.  iterable must be another object that supports
 |  iteration.  The enumerate object yields pairs containing a count (from
 |  start, which defaults to zero) and a value yielded by the iterable argument.
 |  enumerate is useful for obtaining an indexed list:
 |      (0, seq[0]), (1, seq[1]), (2, seq[2]), ...
 |
```

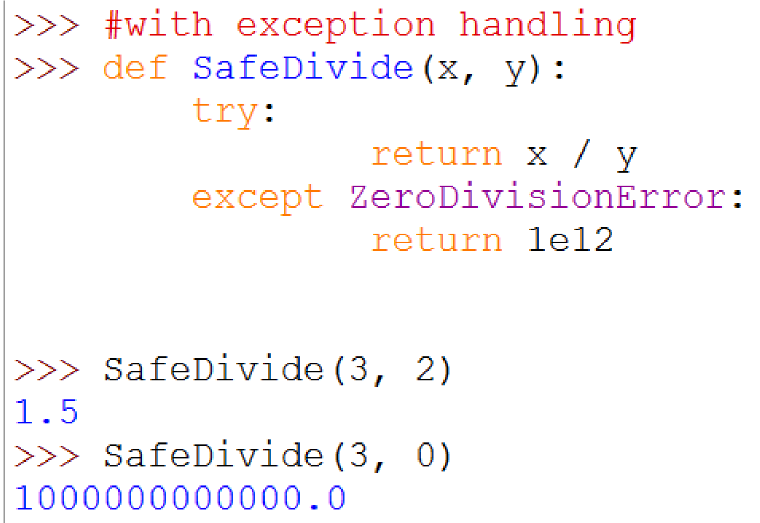


```
>>> pl = ['C', 'C++', 'Java', 'Python']
>>> list(enumerate(pl))
[(0, 'C'), (1, 'C++'), (2, 'Java'), (3, 'Python')]
>>> list(enumerate(pl, start = 3))
[(3, 'C'), (4, 'C++'), (5, 'Java'), (6, 'Python')]
```

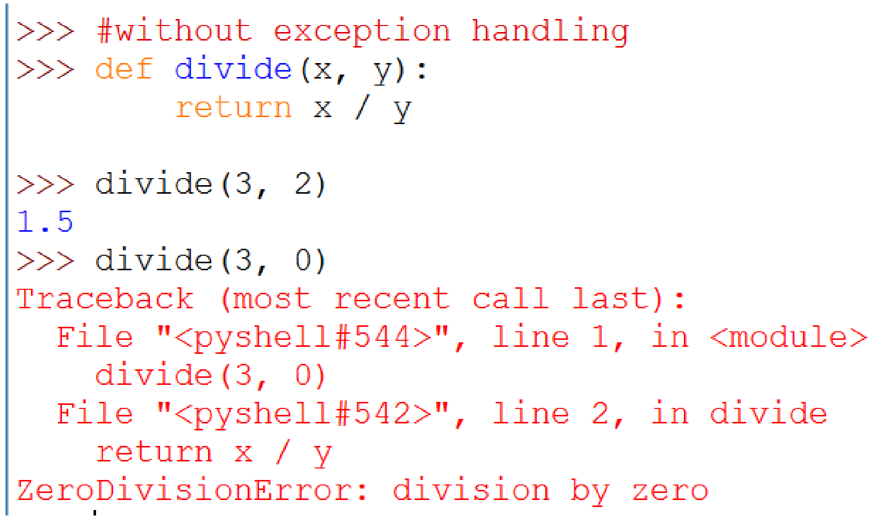
➢ Exceptions

- As programs get bigger and more complex, their behaviour becomes harder to predict.
- Exceptions are a means of breaking out of the normal flow of control of a code block in order to handle errors or other exceptional conditions.
- When an exception is not handled at all, the interpreter terminates execution of the program, or returns to its interactive main loop.
- It is better to use the try statement to specify the exception handlers.
  - try: your code block is put into try statement
  - except: catch the exceptions and handle them
  - finally: specify the clean-up code, which is executed whether an exception occurred or not

```
try:
    code block
except Exception1:
    handle the exception 1
except Exception2:
    handle the exception 2
finally
    suite
```



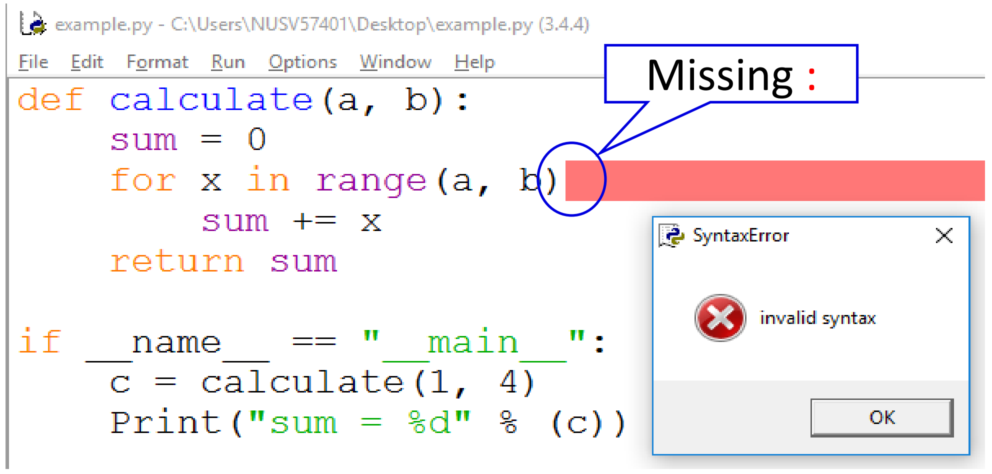
```
>>> #with exception handling
>>> def SafeDivide(x, y):
        try:
                return x / y
        except ZeroDivisionError:
                return 1e12



>>> SafeDivide(3, 2)
1.5
>>> SafeDivide(3, 0)
1000000000000.0
```



```
>>> #without exception handling
>>> def divide(x, y):
        return x / y

>>> divide(3, 2)
1.5
>>> divide(3, 0)
Traceback (most recent call last):
  File "<pyshell#544>", line 1, in <module>
    divide(3, 0)
  File "<pyshell#542>", line 2, in divide
    return x / y
ZeroDivisionError: division by zero
```

Exceptions

- Pay attention to the error information provided by Python
- Trace back to the corresponding line
- Possible error: compile-time error



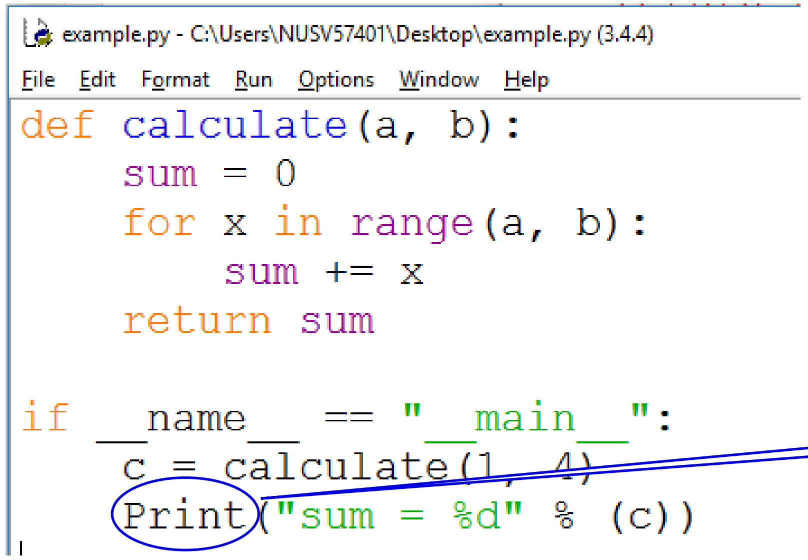
example.py - C:\Users\NUSV57401\Desktop\example.py (3.4.4)

```
def calculate(a, b):
    sum = 0
    for x in range(a, b)
        sum += x
    return sum

if __name__ == "__main__":
    c = calculate(1, 4)
    Print("sum = %d" % (c))
```

SyntaxError

invalid syntax

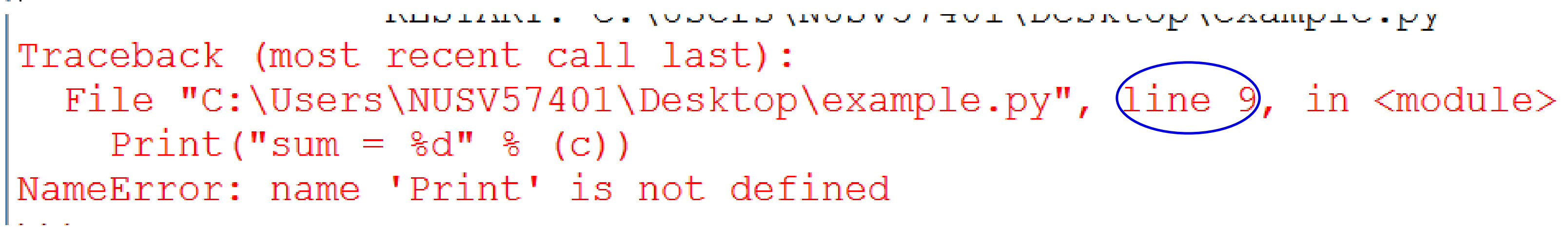
Missing :



example.py - C:\Users\NUSV57401\Desktop\example.py (3.4.4)

```
def calculate(a, b):
    sum = 0
    for x in range(a, b):
        sum += x
    return sum

if __name__ == "__main__":
    c = calculate(1, 4)
    Print("sum = %d" % (c))
```

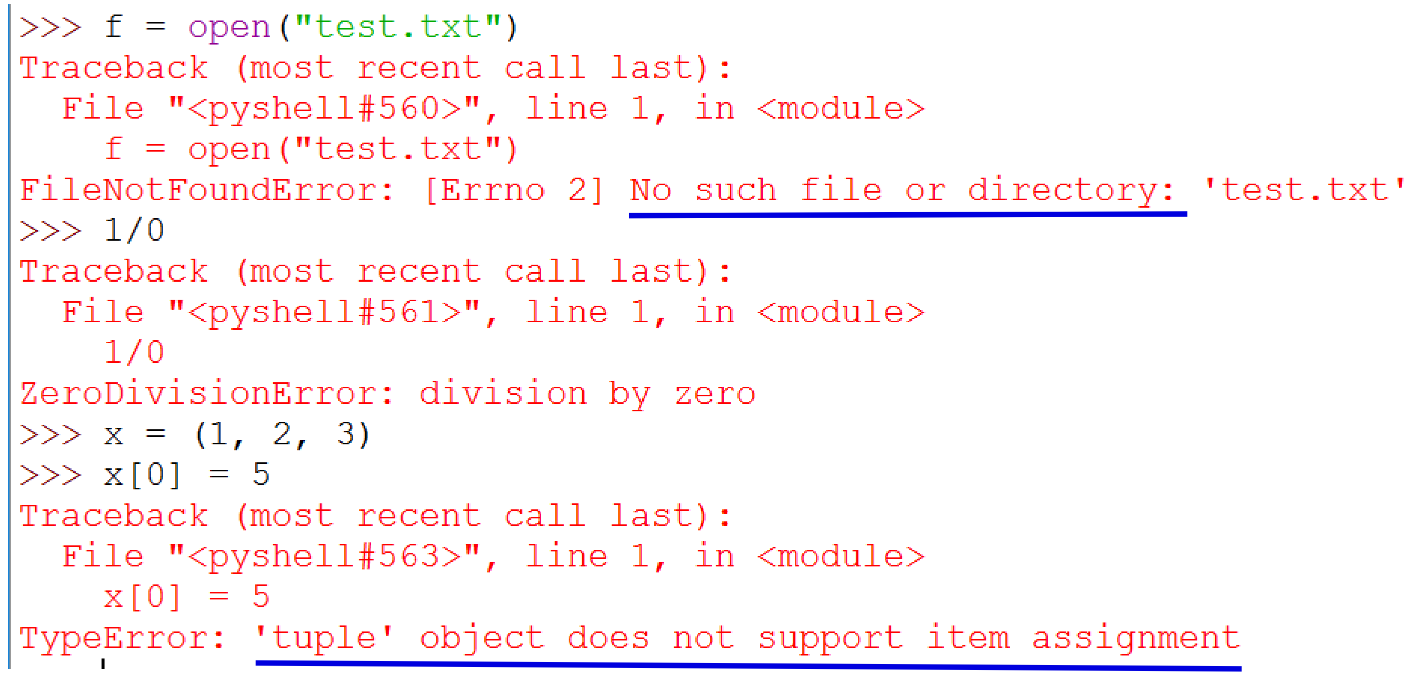
Line 9, should be “print”



```
Traceback (most recent call last):
  File "C:\Users\NUSV57401\Desktop\example.py", line 9, in <module>
    Print("sum = %d" % (c))
NameError: name 'Print' is not defined
```

Exceptions

- Pay attention to the error information provided by Python
- Trace back to the corresponding line
- Possible error: runtime error.
  - Know basics; Be careful to use.



```
>>> f = open("test.txt")
Traceback (most recent call last):
  File "<pyshell#560>", line 1, in <module>
    f = open("test.txt")
FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
>>> 1/0
Traceback (most recent call last):
  File "<pyshell#561>", line 1, in <module>
    1/0
ZeroDivisionError: division by zero
>>> x = (1, 2, 3)
>>> x[0] = 5
Traceback (most recent call last):
  File "<pyshell#563>", line 1, in <module>
    x[0] = 5
TypeError: 'tuple' object does not support item assignment
```

- Logical error. Not detected by compiler. Most hard to debug.
  - Understanding your problem; Read code carefully

➢ Useful packages

- NumPy:
  - powerful numeric computing, including N-dimensional array, matrix operation, linear algebra, Fourier transform, and so on.
  - http://www.numpy.org/
- SciPy:
  - scientific computing, including clustering algorithms, image processing, signal processing, statistical functions.
  - https://www.scipy.org/scipylib
- Matplotlib:
  - powerful 2D plotting library.
  - http://matplotlib.org/

Doing practices is the best way to learn programming language!!!

Thank you, Questions?