Tutorial 2

Variable, Expression, Statement

Object

- Objects are Python’s abstraction for data
- All data in a Python program is representeded by objects or by relations between objects.
- Every objects has an identity, a type and a value.
  - int, float, complex, list, string..
  - Staements
  - Functions
  - Classes
  - ……

Names

- Names (i.e., variables, functions) refer to objects
- Names are introduced by name binding operations

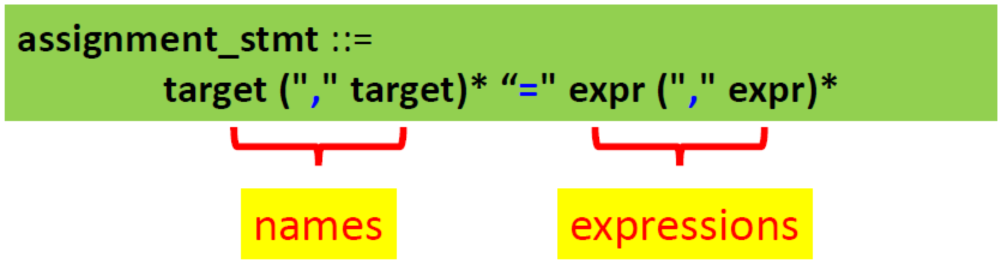
Names

- Names (i.e., variables, functions) refer to objects
- Names are introduced by name binding operations
- Assignment statement **binds** a variable to an object

$$
\begin{array}{l}
\text{assignment\_stmt ::=} \\
\qquad \underbrace{\text{target ("," target)*}}_{\text{names}}\ \text{“=" }\ \underbrace{\text{expr ("," expr)*}}_{\text{expressions}}
\end{array}
$$

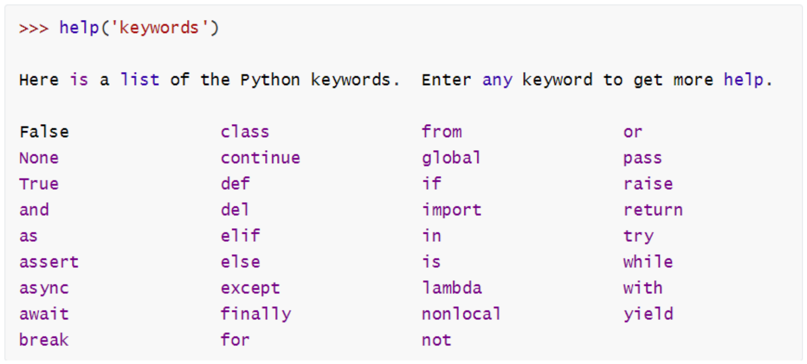
Names

- Names (i.e., variables, functions) refer to objects
- Names are introduced by name binding operations
- Assignment statement **binds** a variable to an object



```
assignment_stmt ::=
        target ("," target)* "=" expr ("," expr)*
```

  - Be aware of the keywords



```
>>> help('keywords')

Here is a list of the Python keywords.  Enter any keyword to get more help.

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

Number

- int
- float
- complex
- ……

Number

- int
- float
  - ways to creat a float:

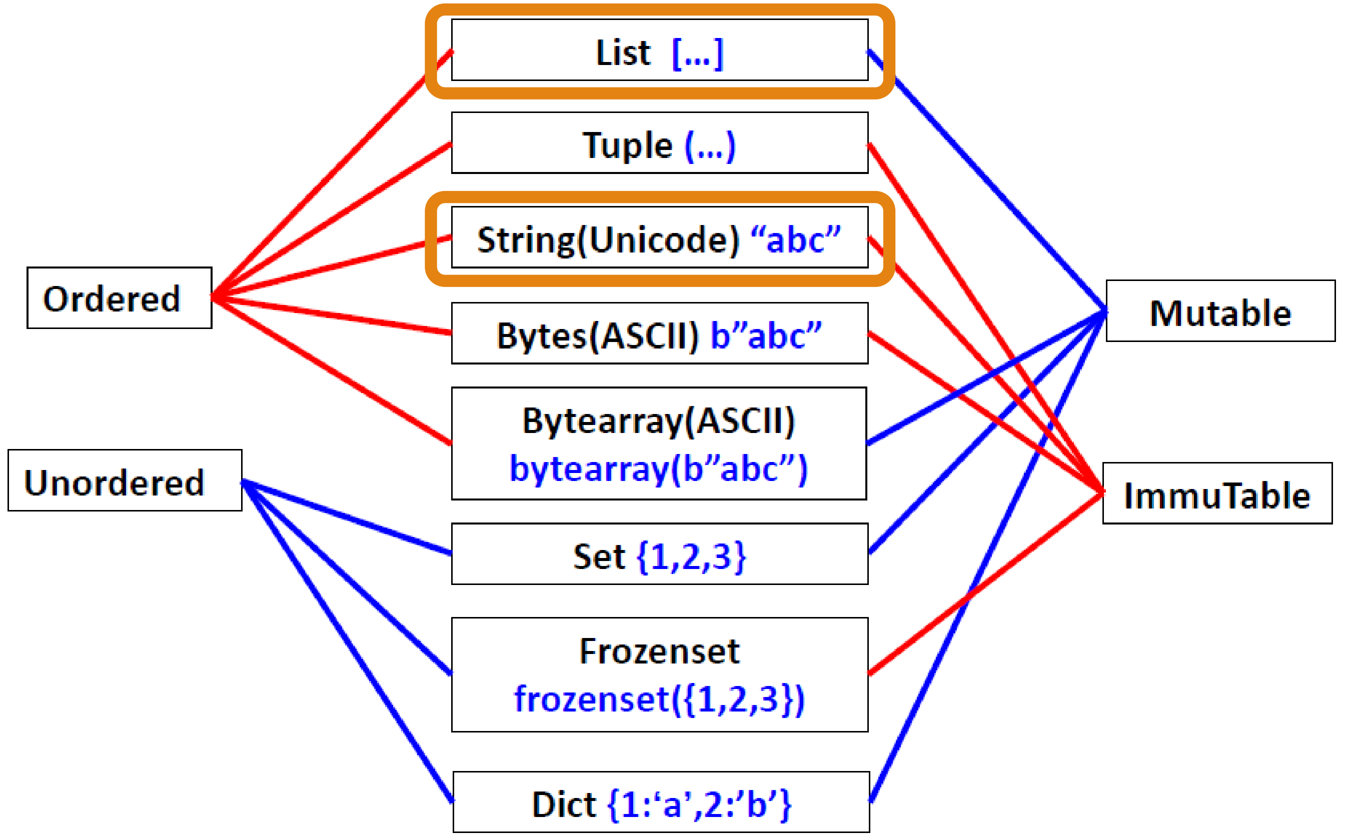
```
>>> (’+1.23’)
1.23
>>> (’  -12345\n’)
-12345.0
>>> (’1e-003’)
0.001
>>> (’+1E6’)
1000000.0
>>> (’-Infinity’)
-inf
```

Number

Truth Value Testing

- Any object can be tested for truth value
- By default, an object is considered true unless its class defines either a __bool__() method that returns **False** or a __len__() method that returns zero
- Here are most of the built-in objects considered false:
  - Constants defined to be false: **None** and **False**
  - Zero of any numeric type: **0, 0.0, 0j, Decimal(0), Fraction(0,1)**
  - Empty sequences and collections: **'', (), [], {}, set(), range(0)**

Overview


List [...]
Tuple (...)
String(Unicode) "abc"
Ordered
Bytes(ASCII) b"abc"
Bytearray(ASCII)
bytearray(b"abc")
Unordered
Set {1,2,3}
Frozenset
frozenset({1,2,3})
Dict {1:'a',2:'b'}
Mutable
ImmuTable

Strings

- `\` can be used to escape quotes:
  - `'`

```
>>> 'I've had dinner.'
SyntaxError: invalid syntax
>>> 'I\'ve had dinner.'    #转义符
"I've had dinner."
```

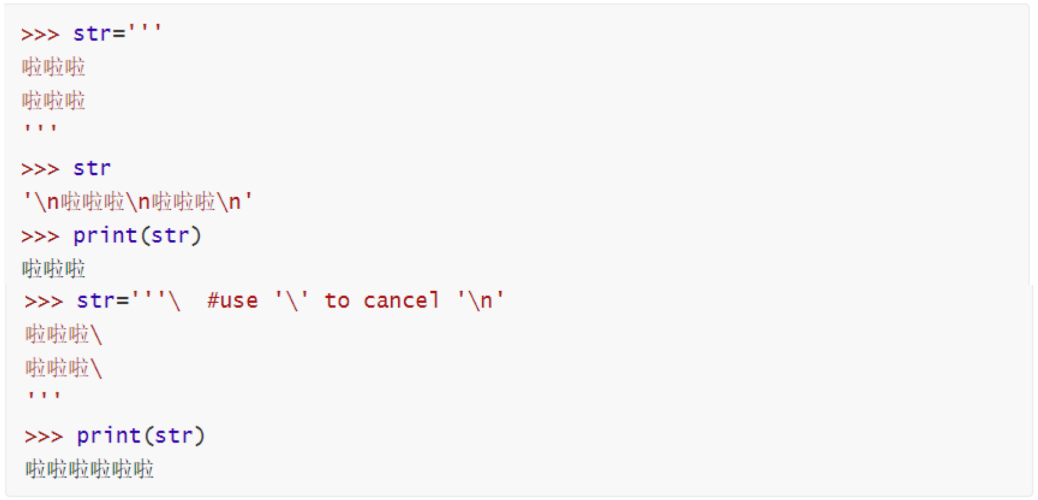
Strings

- `\` can be used to escape quotes:
  - 也可以str=r'C:\now'表示使用原始字符串 (raw string)

```
>>> >>> str=r'C:\now'
>>> str
'C:\\now'
>>> print(str)
C:\now
```

Strings

- 如果希望得到一个跨越多行的字符串，可以使用三重引号字符串



```
>>> str='''
啦啦啦
啦啦啦
'''
>>> str
'\n啦啦啦\n啦啦啦\n'
>>> print(str)
啦啦啦
>>> str='''\  #use '\' to cancel '\n'
啦啦啦\
啦啦啦\
'''
>>> print(str)
啦啦啦啦啦啦
```

Strings

- e.g.1

  Which of the following are valid ways to specify the string literal foo’ bar in Python:

  - “”””foo’bar”””””
  - ‘foo”bar’
  - ‘foo\’bar’
  - ‘foo’bar’
  - “foo’bar”

Strings

- e.g.1

  Which of the following are valid ways to specify the string literal foo’ bar in Python:

  🟧 “””foo’bar”””

  - ‘foo”bar’

  🟧 ‘foo\’bar’

  - ‘foo’bar’

  🟧 “foo’bar”

Strings

- Strings can be *indexed* (subscripted), with the first character having index 0

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
>>> word[-2:]  # characters from the second-last (included) to the end
'on'
>>> word[:2] + word[2:]
'Python'
```

Strings

- Strings can be *indexed* (subscripted), with the first character having index 0
  - Python strings cannot be changed — they are **immutable**. Therefore, assigning to an indexed position in the string results in an error:

```
>>> word[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Strings

- Strings can be *indexed* (subscripted), with the first character having index 0
- Operators: `+`, `*`

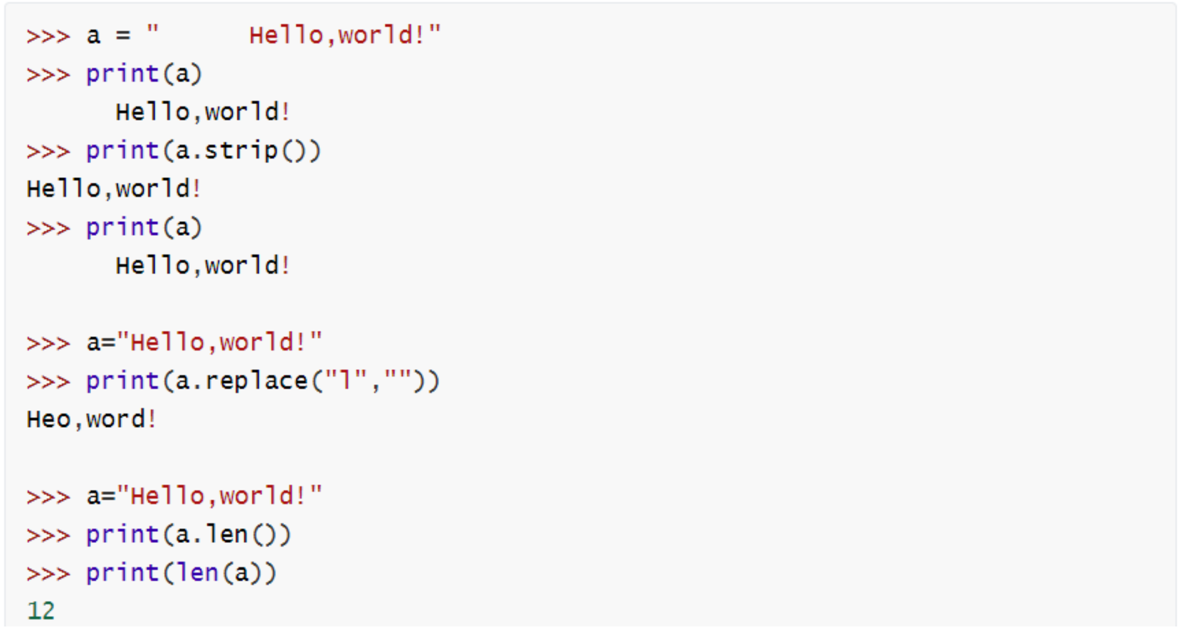
```
>>> 3 * 'un' + 'ium'
'unununium'
```

Strings (search)

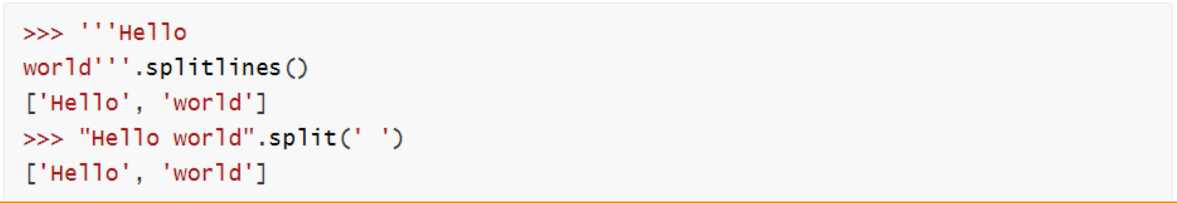
- **s.find(sub[, start[, end]])**
  - return the lowest index in S where substring sub is found such that sub is contained within s[start:end]; Return -1 on failure
- **s.rfind(sub[, start[, end]])**
  - return the highest index in S where substring sub is found such that sub is contained within s[start:end]; Return -1 on failure
- **s.index(sub[, start[, end]])** and **s.rindex(sub[, start[, end]])** are similar to s.find and s.rfind, except that Raises ValueError on failure
- **s.count(sub[, start[, end]])**
  - return the number of nonoverlapping occurrences of substring sub in string s[start:end]

Strings(split & replace)

- Some methods



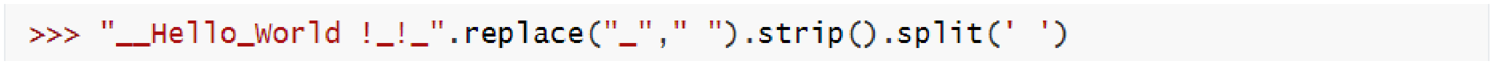
```
>>> a = "      Hello,world!"
>>> print(a)
      Hello,world!
>>> print(a.strip())
Hello,world!
>>> print(a)
      Hello,world!

>>> a="Hello,world!"
>>> print(a.replace("l",""))
Heo,word!

>>> a="Hello,world!"
>>> print(a.len())
>>> print(len(a))
12
```



```
>>> '''Hello
world'''.splitlines()
['Hello', 'world']
>>> "Hello world".split(' ')
['Hello', 'world']
```

Strings

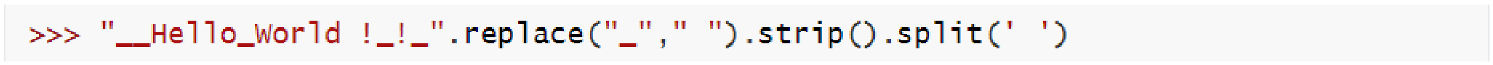
e.g.2 What is the output?



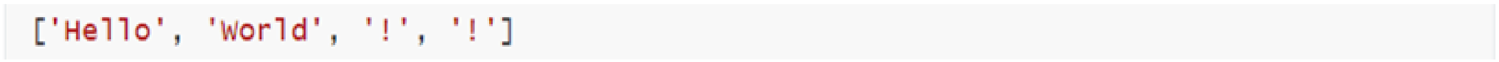
```
>>> "__Hello_World !_!_".replace("_"," ").strip().split(' ')
```

Strings

e.g.2 What is the output?

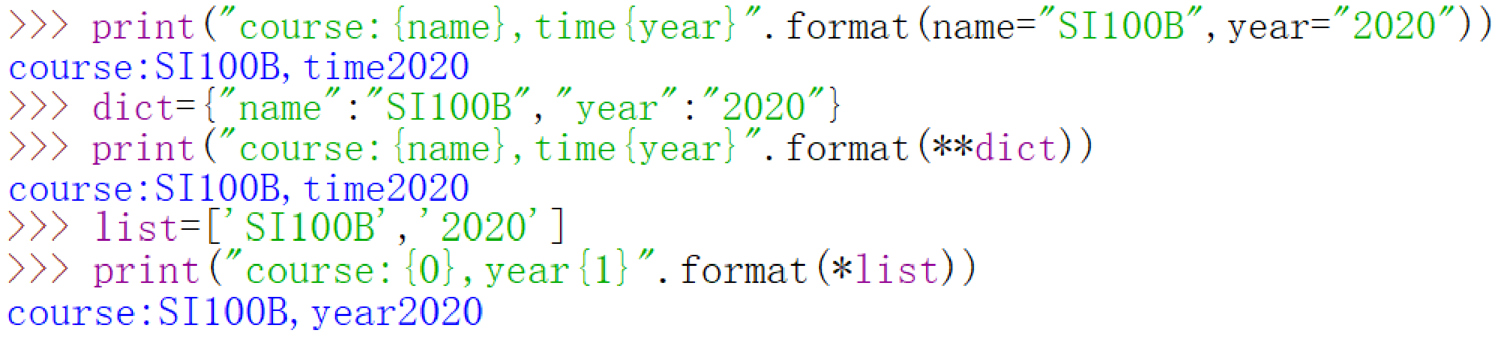


```
>>> "__Hello_World !_!_".replace("_"," ").strip().split(' ')
```



```
['Hello', 'World', '!', '!']
```

Strings--Format String



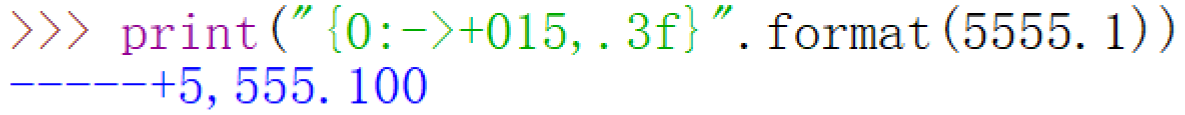
```
>>> print("course:{name},time{year}".format(name="SI100B",year="2020"))
course:SI100B,time2020
>>> dict={"name":"SI100B","year":"2020"}
>>> print("course:{name},time{year}".format(**dict))
course:SI100B,time2020
>>> list=['SI100B','2020']
>>> print("course:{0},year{1}".format(*list))
course:SI100B,year2020
```

Format Specification

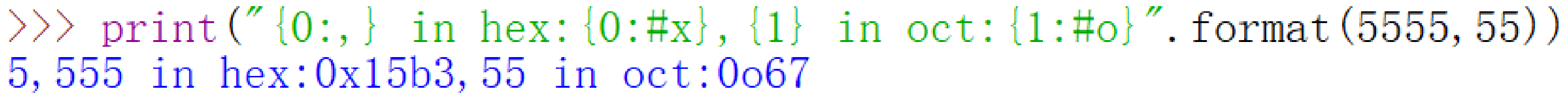
[[fill]align][sign][#][0][width][,|-][.precision][type]

- **fill:** can be any character (default to a space)
- **align:**
  - '<' left-aligned(default)
  - '>':right-aligned
  - '=': fill after sign (only valid for numeric types)
  - '^': centered
- **sign:** only valid for number types
  - '+': a sign for both positive and negative numbers
  - '-': a sign only for negative numbers (default)
  - space: space for positive numbers, a sign for negative numbers
- **#:** '0b', '0o', or '0x' for binary, octal, or hexadecimal
- **,|-:** thousands separator
- **0** : sign-aware zero-padding for numeric types

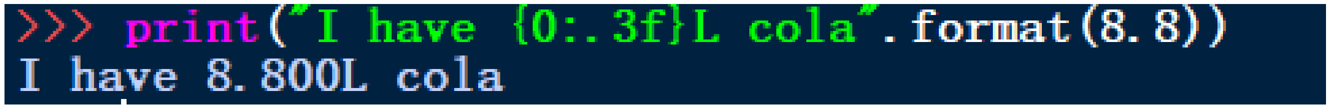
Type	Meaning
's'	String format (default)
'b'	Binary format. Outputs the number in base 2
'c'	Character. Converts the integer to the corresponding unicode character before printing
'd'	Decimal Integer. Outputs the number in base 10 (default)
'o'	Octal format. Outputs the number in base 8
'x'	Hex format. Outputs the number in base 16, using lower-case letters for the digits above 9
'X'	Hex format. Outputs the number in base 16, using upper-case letters for the digits above 9
'n'	Number. This is the same as 'd', except that it uses the current locale setting to insert the appropriate number separator characters



```
>>> print("{0:->+015,.3f}".format(5555.1))
------+5,555.100
```



```
>>> print("{0:,} in hex:{0:#x}, {1} in oct:{1:#o}".format(5555,55))
5,555 in hex:0x15b3,55 in oct:0o67
```



```
>>> print("I have {0:.3f}L cola".format(8.8))
I have 8.800L cola
```

to learn more:https://docs.python.org/3/library/string.html

Strings

See also:

Text Sequence Type — str

- Strings are examples of sequence types, and support the common operations supported by such types.

- String Methods
  - Strings support a large number of methods for basic transformations and searching.
- Formatted string literals
  - String literals that have embedded expressions.
- Format String Syntax
  - Information about string formatting with str.format().
- printf-style String Formatting
  - The old formatting operations invoked when strings are the left operand of the % operator are described in more detail here.

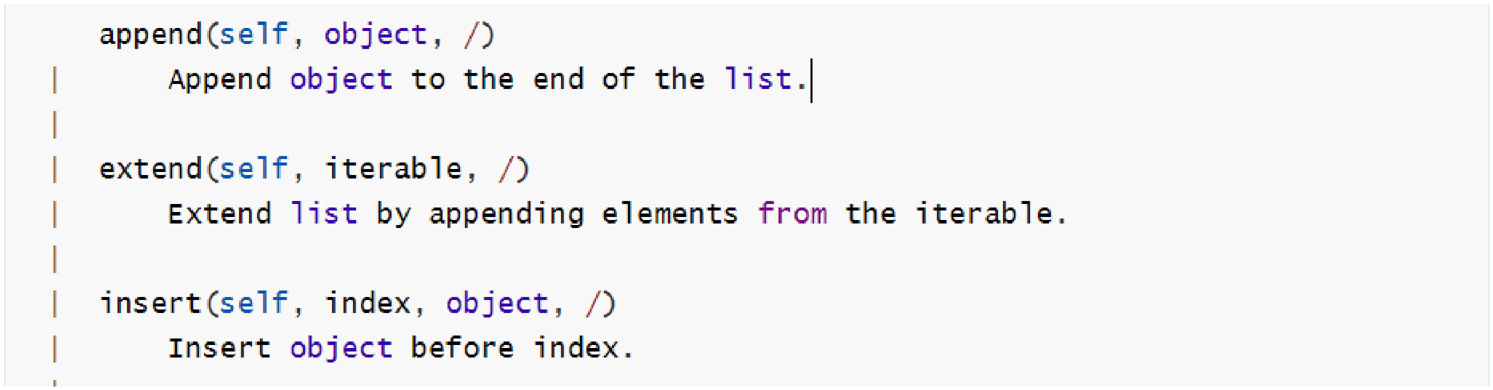
Lists

- Create
  - 1. x = []
  - 2. x = list()
  - 3. x = [1, 2, 3]
  - 4. x = [1, ‘a’, [3, ‘b’], 4]

- From other iterable objects
  - 5. x = list(range(2,10,3))
  - 6. x = list((1,2,3))
  - 7. x = list(“123”)

Lists

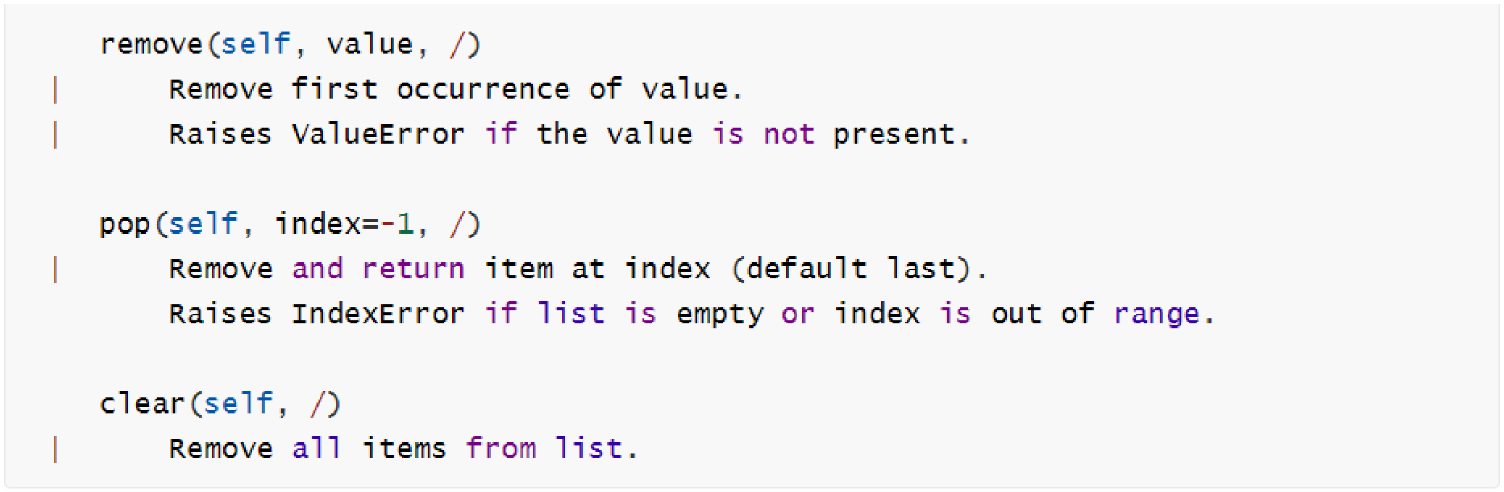
- Methods



```
    append(self, object, /)
 |      Append object to the end of the list.
 |
 |  extend(self, iterable, /)
 |      Extend list by appending elements from the iterable.
 |
 |  insert(self, index, object, /)
 |      Insert object before index.
```

Lists

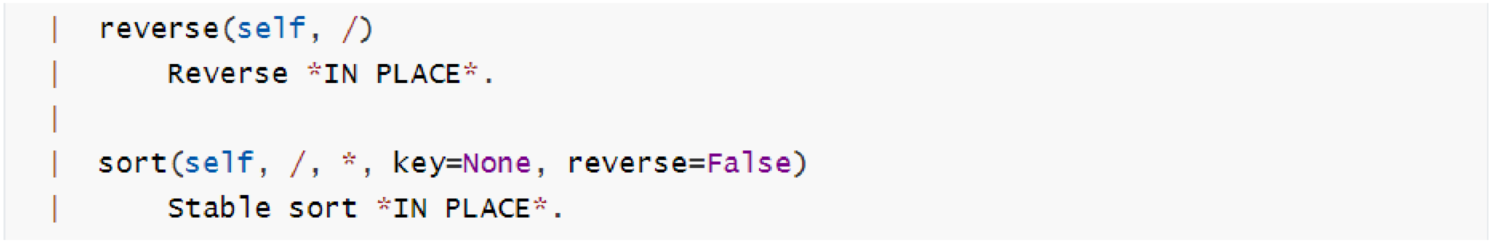
- Methods



```
   remove(self, value, /)
|      Remove first occurrence of value.
|      Raises ValueError if the value is not present.

   pop(self, index=-1, /)
|      Remove and return item at index (default last).
       Raises IndexError if list is empty or index is out of range.

   clear(self, /)
|      Remove all items from list.
```

Lists

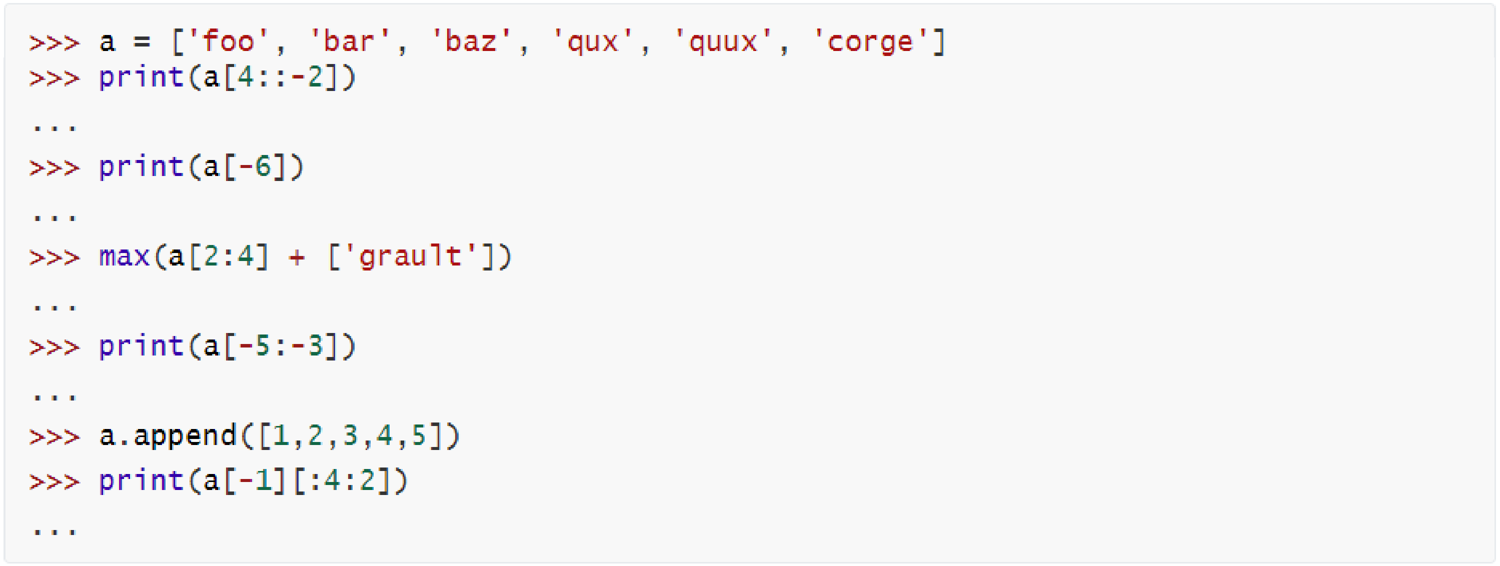
- Methods



```
 |  reverse(self, /)
 |      Reverse *IN PLACE*.
 |
 |  sort(self, /, *, key=None, reverse=False)
 |      Stable sort *IN PLACE*.
```

Lists

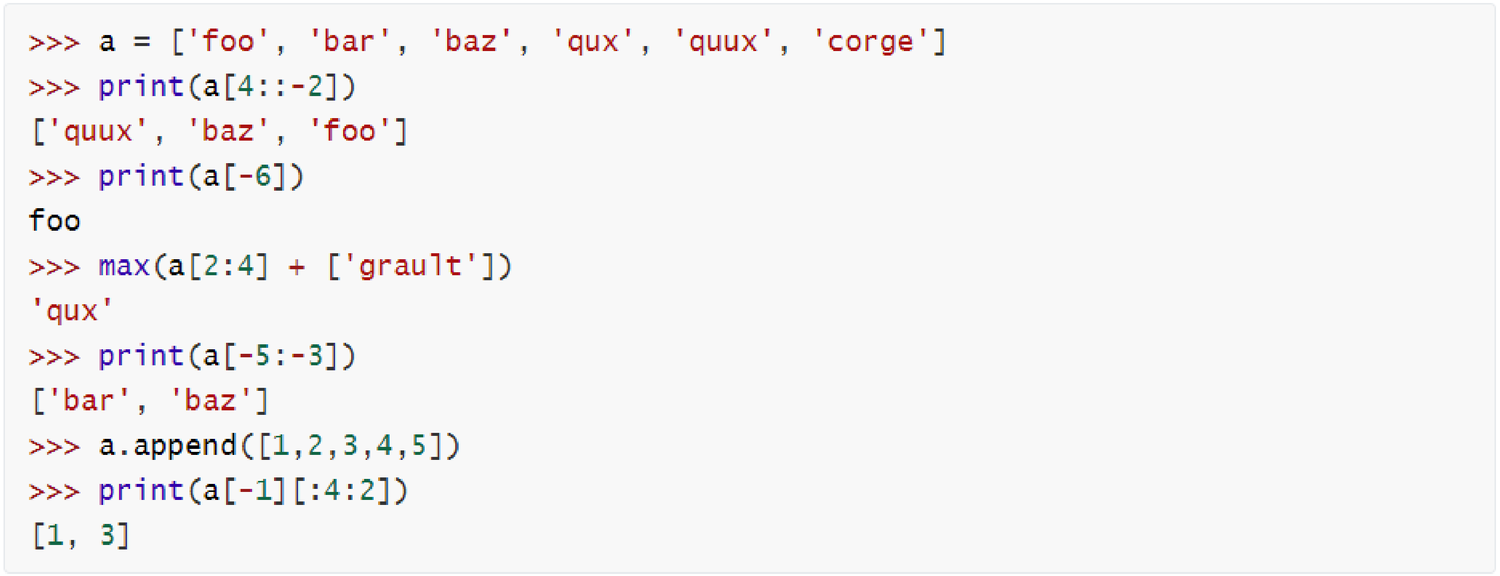
- e.g.3 What is the output?



```
>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
>>> print(a[4::-2])
...
>>> print(a[-6])
...
>>> max(a[2:4] + ['grault'])
...
>>> print(a[-5:-3])
...
>>> a.append([1,2,3,4,5])
>>> print(a[-1][:4:2])
...
```

Lists

- e.g.3 What is the output?



```
>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
>>> print(a[4::-2])
['quux', 'baz', 'foo']
>>> print(a[-6])
foo
>>> max(a[2:4] + ['grault'])
'qux'
>>> print(a[-5:-3])
['bar', 'baz']
>>> a.append([1,2,3,4,5])
>>> print(a[-1][:4:2])
[1, 3]
```

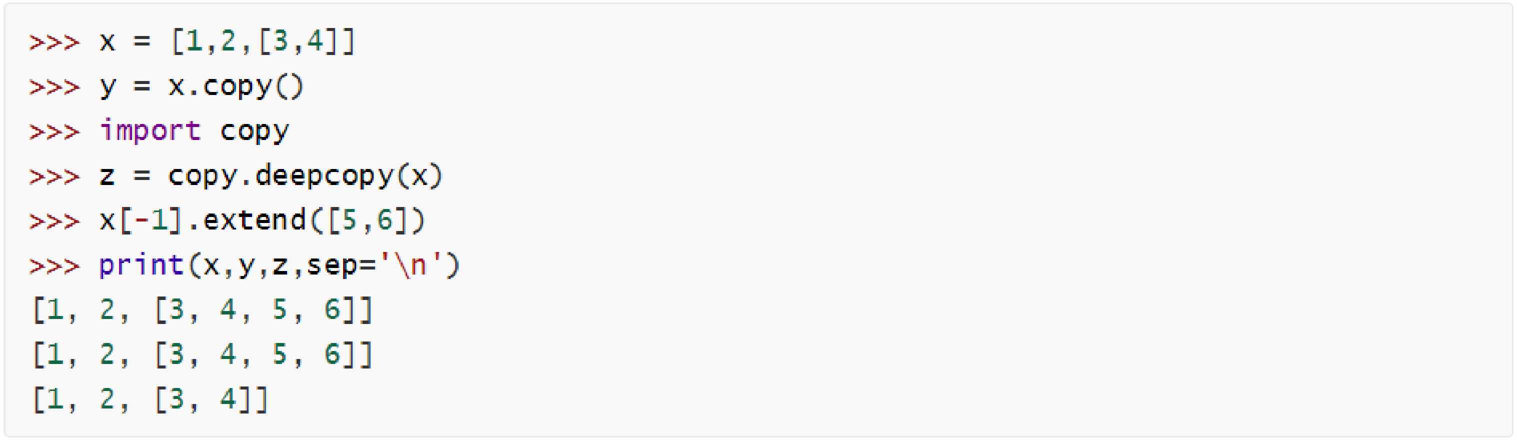
Lists

- e.g. 4 About copy
- What is the output?

```
>>> x = [1,2,[3,4]]
>>> y = x.copy()
>>> import copy
>>> z = copy.deepcopy(x)
>>> x[-1].extend([5,6])
>>> print(x,y,z,sep='\n')
......
```

Lists

- e.g. 4 About copy
- What is the output?



```
>>> x = [1,2,[3,4]]
>>> y = x.copy()
>>> import copy
>>> z = copy.deepcopy(x)
>>> x[-1].extend([5,6])
>>> print(x,y,z,sep='\n')
[1, 2, [3, 4, 5, 6]]
[1, 2, [3, 4, 5, 6]]
[1, 2, [3, 4]]
```

Lists

More on lists

https://docs.python.org/3/tutorial/datastructures.html#more-on-list

Problem

- 有一个小程序，包括以下步骤：

  1. 声明一些int变量并赋初值；

  2. 有一个if block，语法为：if (expression) {statements} fi；

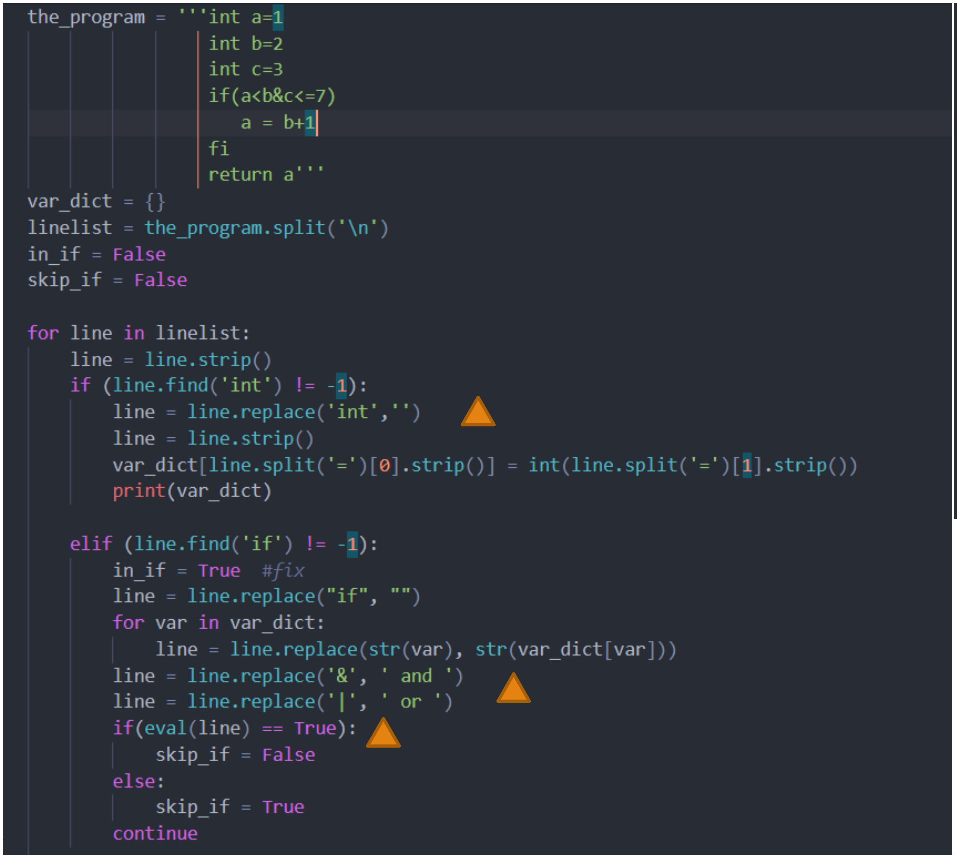
  3. 返回一个之前声明的整数的值

- e.g.

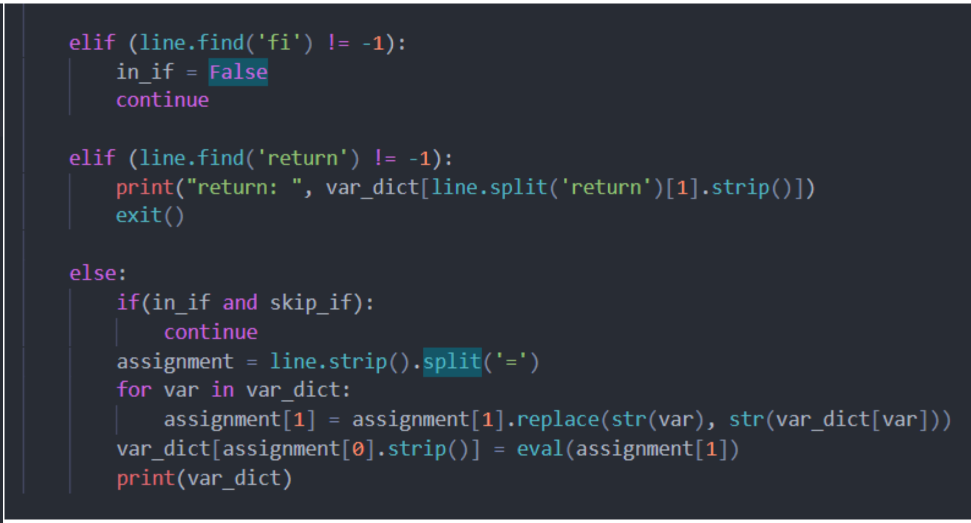
```
int a=1
int b=2
int c=3
    if(a<b&c<1)
        a = b+1
    fi
return a
```

- 输入这个小程序， 输出其返回值。

Problem



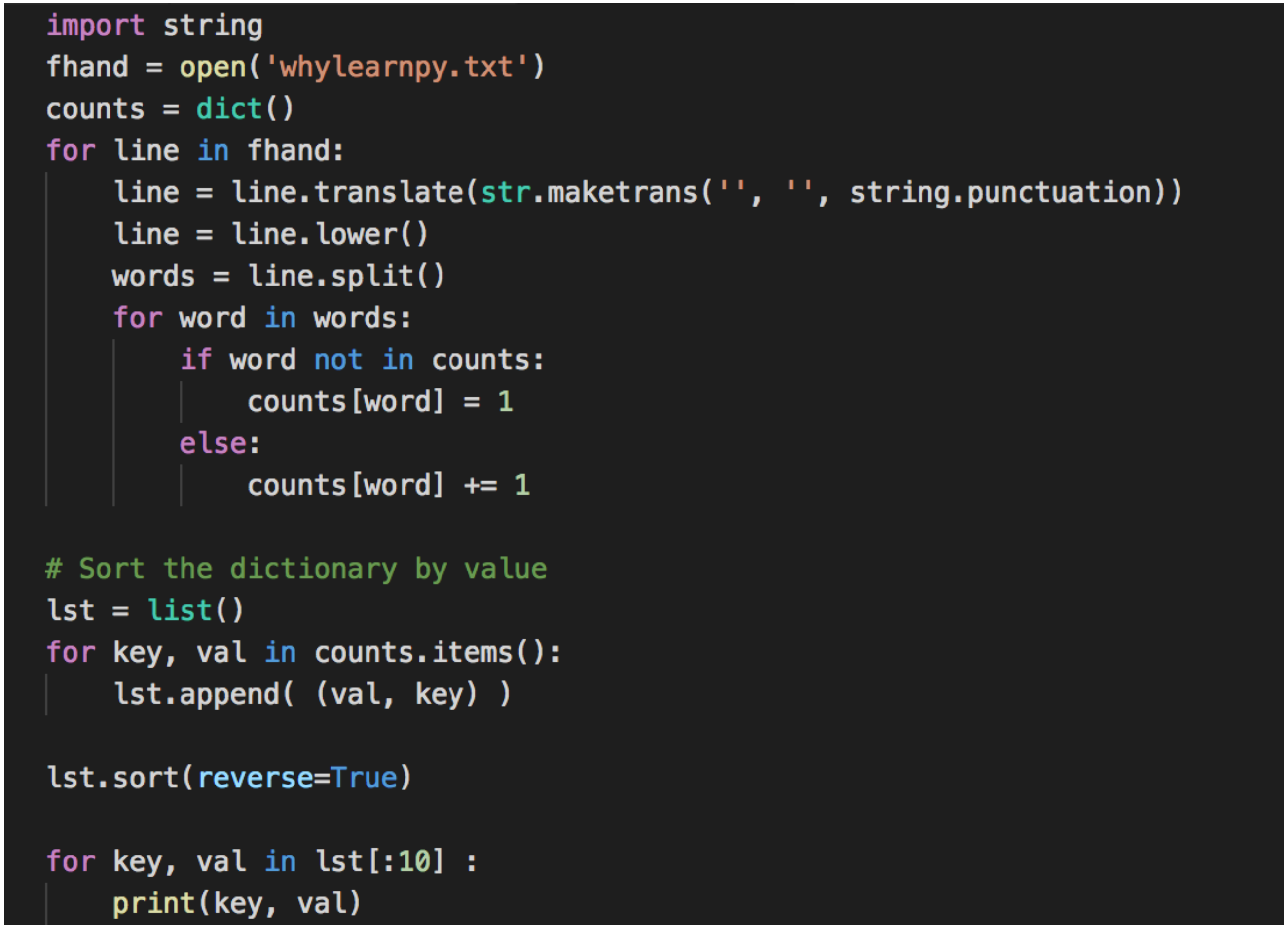
```
the_program = '''int a=1
                 int b=2
                 int c=3
                 if(a<b&c<=7)
                    a = b+1
                 fi
                 return a'''
var_dict = {}
linelist = the_program.split('\n')
in_if = False
skip_if = False

for line in linelist:
    line = line.strip()
    if (line.find('int') != -1):
        line = line.replace('int','')
        line = line.strip()
        var_dict[line.split('=')[0].strip()] = int(line.split('=')[1].strip())
        print(var_dict)

    elif (line.find('if') != -1):
        in_if = True  #fix
        line = line.replace("if", "")
        for var in var_dict:
            line = line.replace(str(var), str(var_dict[var]))
        line = line.replace('&', ' and ')
        line = line.replace('|', ' or ')
        if(eval(line) == True):
            skip_if = False
        else:
            skip_if = True
        continue
```



```
    elif (line.find('fi') != -1):
        in_if = False
        continue

    elif (line.find('return') != -1):
        print("return: ", var_dict[line.split('return')[1].strip()])
        exit()

    else:
        if(in_if and skip_if):
            continue
        assignment = line.strip().split('=')
        for var in var_dict:
            assignment[1] = assignment[1].replace(str(var), str(var_dict[var]))
        var_dict[assignment[0].strip()] = eval(assignment[1])
        print(var_dict)
```

THE MOST COMMON WORDS example1 20min

- 读取 whylearnpy.txt
- 统计哪些单词出现了最多次
- output：前十 （次数&单词）



```python
import string
fhand = open('whylearnpy.txt')
counts = dict()
for line in fhand:
    line = line.translate(str.maketrans('', '', string.punctuation))
    line = line.lower()
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

# Sort the dictionary by value
lst = list()
for key, val in counts.items():
    lst.append( (val, key) )

lst.sort(reverse=True)

for key, val in lst[:10] :
    print(key, val)
```

https://docs.python.org/3.7/index.html