# 大学计算机入门课

**Class #7**

[教学目的]

- More on while loops
- Learn new data types --- set and Dictionary

[课程大纲]

- Part 1: more on while loops
- Part 2: set
- Part 3: dictionary

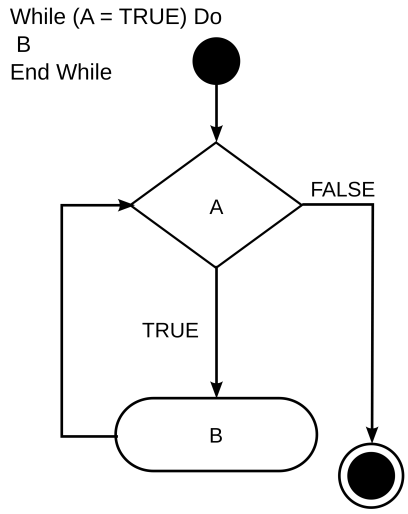
## part 1: more on while loops

- recall while loops

  A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition.

  The while loop can be thought of as a repeating if statement.

- Graphically:




- More functions
  - Consider a function that returns the the index of first occurrence of “stop”, If there is not such string “stop” in the list, let’s return -1 instead.

```
def stop_index(l):
    """ (list) -> int

    Return the index of first occurrence of "stop", if there is not such string
    "stop" in the list, return -1.

    >>> stop_index(["hello", 1, "wow", "stop", 2, 3])
    3
    >>> stop_index(["hello", 1, "wow"])
    -1
    """
```

**Case 1:**

Firstly, let's consider the list ["hello", 1, "wow", "stop", 2, 3]

| "hello" | 1 | "wow" | "stop" | 2 | 3 |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

所以我们 return 3 here。

First of all,

Initialize index = 0, then move index one by one until we found l[index] == "stop"

So, when we stop, index record to the value 3.

Return index.

So our loop should like:

**Case 1 code:**

```
Index = 0
While l[index] != "stop":
        Index += 1
Return index
```

**Case 2:**

let's consider the list ["hello", 1, "wow"]

| "hello" | 1 | "wow" |
|---|---|---|
| 0 | 1 | 2 |

We should return -1 here.

But if we use to case 1 code here, what problem will occur?

INFINITE LOOPS!! Never stop, since l[index] != "stop" always true.

How to fix it?

ADD MORE RESITRICTION!

We will stop the while loop if we reach to the end of this list.

Case 2 code:

```
index = 0
while index < len(l) and l[index] != "stop":
```

```
        Index += 1
if index == len(l):
        return  -1
return index
```

We have complete our code!

```
def stop_index(l):
    """ (list) -> int

    Return the index of first occurrence of "stop", if there is not such stri
    "stop" in the list, return -1.

    >>> stop_index(["hello", 1, "wow", "stop", 2, 3])
    3
    >>> stop_index(["hello", 1, "wow"])
    -1
    """
    index = 0
    while index < len(l) and l[index] != "stop":
        index += 1
    if index == len(l):
        return -1
    return index
```

- Now, we also should know how to read a while loop! More exercises!

```
def mystery():
    i = 0
    numbers = []
    while i < 6:
        print("At the top i is " + str(i))
        numbers.append(i)

        i = i + 1
        print("Numbers now: ", numbers)
        print("At the bottom i is " + str(i))
    print("The numbers: ")
    for num in numbers:
        print(num)
```

Now,consider after excute
>>> mystery()
what will display in the python shell?

```
>>> mystery()
    At the top i is 0
    Numbers now:  [0]
    At the bottom i is 1
    At the top i is 1
    Numbers now:  [0, 1]
    At the bottom i is 2
    At the top i is 2
    Numbers now:  [0, 1, 2]
    At the bottom i is 3
    At the top i is 3
    Numbers now:  [0, 1, 2, 3]
    At the bottom i is 4
    At the top i is 4
    Numbers now:  [0, 1, 2, 3, 4]
    At the bottom i is 5
    At the top i is 5
    Numbers now:  [0, 1, 2, 3, 4, 5]
    At the bottom i is 6
    The numbers:
    0
    1
    2
    3
    4
    5
>>>
```

经过阅读和写 while function，我们是不是对 while function 更加熟悉了呢？

- 所以现在我们再来尝试一个 while function。

```
def clean_up_end_digit(lst):
    """(list) -> None

    Modify the original lst such that clean up all ends digit, and print out
    the item poped in the list.

    >>> a = ["hello", 1, 3, 5]
    >>> clean_up_end_digit(a)
    5
    3
    1
    >>> a
    ["hello"]
    >>> a = ["hello", 1, 3, 5, "bye"]
    >>> clean_up_end_digit(a)
    >>> a
    ["hello", 1, 3, 5, "bye"]
    """
```

try to complete it within two lines of code!

```
def clean_up_end_digit(lst):
    """(list) -> None

    Modify the original lst such that clean up all ends digit, and print out
    the item poped in the list.

    >>> a = ["hello", 1, 3, 5]
    >>> clean_up_end_digit(a)
    5
    3
    1
    >>> a
    ['hello']
    >>> a = ["hello", 1, 3, 5, "bye"]
    >>> clean_up_end_digit(a)
    >>> a
    ['hello', 1, 3, 5, 'bye']
    """
    while len(lst) != 0 and isinstance(lst[-1], int):
        print(lst.pop())
```

## part 2: new data type --- set

- what is it?
  A set is an unordered collection of distinct items.
- Python has a type called set that allows us to store mutable collections of unordered, distinct items. (Remember that a mutable object means one that you can modify.)
- Notation: { }

```
>>> s = {1,2,3}
>>> type(s)
    <class 'set'>
>>>
```

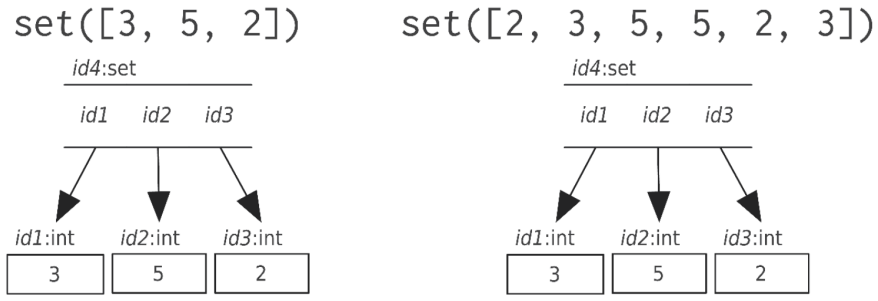
- converting list to set.

```
>>> set([1,2,3])
    {1, 2, 3}
>>>
```

- no duplicates allowed!

```
>>> {2,3,2,5}
    {2, 3, 5}
>>>
```

Because duplicates aren't allowed, only one of the 2s appears in the set:



- set operations

| Method | Description |
|---|---|
| S.add(v) | Adds item v to a set S—this has no effect if v is already in S. |
| S.clear() | Removes all items from set S |
| S.difference(other) | Returns a set with items that occur in set S but not in set other |
| S.intersection(other) | Returns a set with items that occur both in sets S and other |
| S.issubset(other) | Returns True if and only if all of set S's items are also in set other |
| S.issuperset(other) | Returns True if and only if set S contains all of set other's items |
| S.remove(v) | Removes item v from set S |
| S.symmetric_difference(other) | Returns a set with items that are in exactly one of sets S and other—any items that are in both sets are *not* included in the result. |
| S.union(other) | Returns a set with items that are either in set S or other (or in both) |

- method call & operator

| Method Call | Operator |
|---|---|
| set1.difference(set2) | set1 - set2 |
| set1.intersection(set2) | set1 & set2 |
| set1.issubset(set2) | set1 <= set2 |
| set1.issuperset(set2) | set1 >= set2 |
| set1.union(set2) | set1 \| set2 |
| set1.symmetric_difference(set2) | set1 ^ set2 |

- how to initialize an empty set?
  Use: >>> a = set()
  Not: {}
- note!
  most of the set method does not modified the original set!

```
>>> a = {1,2,3}
>>> b = {1}
>>> a.intersection(b)
    {1}
>>> a
    {1, 2, 3}
>>> b
    {1}
```

- set don't have index!  But len(a) still work

```
>>> a = {1,2,3}
>>> len(a)
3
>>>
```

- The equality of two sets only depend on the item inside, not the position.

```
>>> {1,2,3} == {3,1,2}
True
>>>
```

This is different from the list,
Two list are equal iff each index has the same value

```
>>> [1,2,3] == [3,1,2]
False
>>>
```

所以，list 更注重顺序，而 set 在意的只是里面有什么元素。

- The difference between list and set.

| | List | Set |
|---|---|---|
| Notation | [ 1, 2, 3] | { 1, 2, 3} |
| Initialize the empty | >>> a = [] | >>> a = set() |
| Modified | Most of the list method modified the original list | Most of the set method doesn't modified the original list. |
| Have Index? | Yes | no |
| Equality | Two list are equal iff each index has the same value | The equality of two sets only depend on the item inside, not the position |

- Function about set

```
def set_element(st):
    """(str) -> set

    Return a set of alpha that occurs in the string st.

    >>> set_element("hello")
    {'h', 'e', 'l', 'o'}
    """
```

Function with body:

```
def set_element(st):
    """(str) -> set

    Return a set of alpha that occurs in the string st.

    >>> set_element("hello")
    {'h', 'e', 'l', 'o'}
    """
    result = set()
    for s in st:
        result.add(s)
    return result
```

Note: we need pay attention to the example of the function that return a set.
We may have error if we using doctest to test.

```
Failed example:
    set_element("hello")
Expected:
    {'h', 'e', 'l', 'o'}
Got:
    {'o', 'l', 'h', 'e'}
**********************************************************************
```

we know they are the same, but it may print out in different position,
so we may write the example as:

```
def set_element(st):
    """(str) -> set

    Return a set of alpha that occurs in the string st.

    >>> result = set_element("hello")
    >>> result == {'h', 'e', 'l', 'o'}
    True
    """
    result = set()
    for s in st:
        result.add(s)
    return result
```

in this way, we will pass the doctest.

## Part 3: new data type – dictionary

- What is it?
  The dictionary has a key and a value in pairs.
  Dic = {key: value}

```
>>> tel = {'Annie': 123456, 'Cherry': 34566}
>>> tel
{'Annie': 123456, 'Cherry': 34566}
>>>
```

- What things cannot be the key?
  Anything that can be modified cannot be the key.
  Eg: list, set…

  But String、number all can be the key
- Is there any restriction about the value?
  No, anything can be the value.
- Dictionary 也不注重顺序

```
>>> {1: 2, 3: 4} == {3: 4, 1: 2}
True
>>>
```

  但是 pair（key： value）必须要对应

```
>>> {1: 2, 3: 4} == {3: 2, 1: 4}
False
>>>
```

- Dictionary 也没有 duplicate 的 key

```
>>> tel = {'Annie': 123456, 'Annie': 34566}
>>> tel
{'Annie': 34566}
>>>
```

  value 只选择记忆。
- Don’t have index but have length

```
>>> len({1:2})
1
>>>
```

  # the length of a dictionary is the total number of the key in this dictionary.
- How to initialize a new dictionary?
  >>> a = { }
- Dictionary can be modified
  - 如果我们想要改已有 key 的 value

```
>>> tel = {'Annie': 123456, 'Cherry': 34566}
>>> tel['Annie'] = 213
>>> tel
{'Annie': 213, 'Cherry': 34566}
>>>
```

如果一个 key 的 value 时 mutable 的话，我们也可以用 value 的 method 来改变它：

```
>>> mark = {"level_1": [1,2,3], "level_2": [4,5,6]}
>>> mark["level_1"].append(7)
>>> mark
{'level_1': [1, 2, 3, 7], 'level_2': [4, 5, 6]}
>>>
```

- 想要加入一个新的 key：value

```
>>> tel = {'Annie': 123456, 'Cherry': 34566}
>>> tel['Alex'] = 321
>>> tel
{'Annie': 123456, 'Alex': 321, 'Cherry': 34566}
>>>
```

- 我们无法只加入一个 key， 因为 key 与 value 必须是一对同时存在的
- 我们也无法修改已有的 key，只能通过 method 去删除一个 key

- dictionary operation

| Method | Description |
|---|---|
| D.clear() | Removes all key/value pairs from dictionary D. |
| D.get(k) | Returns the value associated with key k, or None if the key isn't present. (Usually you'll want to use D[k] instead.) |
| D.get(k, v) | Returns the value associated with key k, or a default value v if the key isn't present. |
| D.keys() | Returns dictionary D's keys as a set-like object—entries are guaranteed to be unique. |
| D.items() | Returns dictionary D's (key, value) pairs as set-like objects. |
| D.pop(k) | Removes key k from dictionary D and returns the value that was associated with k—if k isn't in D, an error is raised. |
| D.pop(k, v) | Removes key k from dictionary D and returns the value that was associated with k; if k isn't in D , returns v. |
| D.setdefault(k) | Returns the value associated with key k in D. |
| D.setdefault(k, v) | Returns the value associated with key k in D; if k isn't a key in D, adds the key k with the value v to D and returns v. |
| D.values() | Returns dictionary D's values as a list-like object—entries may or may not be unique. |
| D.update(other) | Updates dictionary D with the contents of dictionary other; for each key in other, if it is also a key in D, replaces that key in D's value with the value from other; for each key in other, if that key isn't in D, adds that key/value pair to D. |

- compare the data type that we learned

| Collection | Mutable? | Ordered? | Use When… |
|---|---|---|---|
| str | No | Yes | You want to keep track of text. |
| list | Yes | Yes | You want to keep track of an ordered sequence that you want to update. |
| set | Yes | No | You want to keep track of values, but order doesn't matter, and you don't want to keep duplicates. The values must be immutable. |
| dictionary | Yes | No | You want to keep a mapping of keys to values. The keys must be immutable. |

**functions for dictionary**

- pair_name_tel

  note: pay attention to the example as well

```
def pair_name_age(l1, l2):
    """(list[str], list[int]) -> dic

    Precondition: len(l1) == len(l2)

    Return the dictionary with name int l1 as the key and age in l2 as the value.

    >>> result = pair_name_age(["Annie", "Cherry", "Alex"], [18, 21, 22])
    >>> result == {'Annie': 18, 'Cherry': 21, 'Alex': 22}
    True
    """
```

consider the example firstly:

| Index | 0 | 1 | 2 |
|---|---|---|---|
| L1 | 'Annie' | 'Cherry' | 'Alex' |
| L2 | 18 | 21 | 22 |

So, in the dictionary, dic_result[l1[index]] = l2[index]

Then, we can complete the function body.

```
def pair_name_age(l1, l2):
    """(list[str], list[int]) -> dic

    Precondition: len(l1) == len(l2)

    Return the dictionary with name int l1 as the key and age in l2 as the value.

    >>> result = pair_name_age(["Annie", "Cherry", "Alex"], [18, 21, 22])
    >>> result == {'Annie': 18, 'Cherry': 21, 'Alex': 22}
    True
    """
    dic_result  = {}
    i = 0
    for i in range(len(l1)):
        dic_result[l1[i]] = l2[i]

    return dic_result
```

- for loops for dictionary
  we can use for loop to access each key in the dictionary

```
>>> mark = {"level_1": [1,2,3], "level_2": [4,5,6], "level_3": [7,8,9]}
>>> for key in mark:
...     print(key)
...
level_3
level_1
level_2
>>>
```

consider the function, class_average(d) that returns the average mark for the student in this class.

```
def class_average(d):
    """(dic) -> float

    Return the class average of the value in each key in d.

    >>> class_average({'A': 96, 'B': 89, 'C': 67, 'D': 100})
    88.0
    """
```

complete the function body.

```
def class_average(d):
    """(dic) -> float

    Return the class average of the value in each key in d.

    >>> class_average({'A': 96, 'B': 89, 'C': 67, 'D': 100})
    88.0
    """
    total_mark = 0
    for student in d:
        total_mark += d[student]
    return total_mark/len(d)
```