المحاضرة الأولى

Basics of Python

Sequence types :Tuples , Lists, and Strings

Sequence Types

1.Tuple

- A simple ***immutable ordered*** sequence of items
- Immutable: a tuple cannot be modified once created....
- Items can be of mixed types, including collection types

2. Strings

immutable

3.List

- ***mutable*** ordered sequence of items of mixed types

List

```
# declare a list of numbers
numbers = [1,3,5,7,9,55]
print(numbers)
List=[“a”,5,3.4,6,[5,8]]
print(numbers[0])            # will output 1
print(numbers in range [1:4])      # [3,5,7]
numbers.append(10)
print(numbers)               #  = [1,3,5,7,9,55,10]
numbers.insert(1,55)
print(numbers)               #  = [1,55,3,5,7,9,55,10]
numbers.remove(3)          #  = [1,55, 5,7,9,55,10]
```

Sequence Types

- The three sequence types (tuples, strings, and lists) share much of the same syntax and functionality.
- Tuples are defined using parentheses (and commas).

```
>>> tu= (23, 'abc', 4.56, (2,3), 'def')

>>> li= ["abc", 34, 4.34, 23]

>>> st= "Hello World"
>>> st= 'Hello World'
>> >st= """This is a multi-line
string that uses triple quotes."""
```

Operations on Lists

```
>>> li = [1, 11, 3, 4, 5]
>>> li.append('a')# Note the method syntax
>>> li
   [1, 11, 3, 4, 5, 'a']
>>> li.insert(2, 'i')
>>> li
    [1, 11, 'i', 3, 4, 5, 'a']
```

```
>>> li = ['a', 'b', 'c', 'b']
>>> li.index('b') # index of first occurrence*
1
*more complex forms exist
>>> li.count('b') # number of occurrences
2
>>> li.remove('b') # remove first occurrence
>>> li
['a', 'c', 'b']
```

For loop

numbers = [1,3,5,7,9,55]

- *for y in numbers:*

  print(numbers)

- *for x in range(10):*

  print(x)

- *for x in range(2,10,2):*

  print(x) # will output 2,4,6,8

- *for x in range(10):*

  if x == 6:

  break

  print(x) # will output 0,1,2,3,4,5

while Loops

```
>>> x = 3
>> >while x < 5:
        print x,"still in the loop"
        x = x + 1
3 still in the loop
4 still in the loop
>>>x = 6
>>>while x < 5:
        print x,"still in the loop"
```

Break and continue

- **You can use the keyword *break* inside a loop to leave the *while* loop entirely.**
- **You can use the keyword *continue* inside a loop to stop processing the current iteration of the loop and immediately go on to the next one.**

Lists: Mutable

```
>>> li = ['abc', 23, 4.34, 23]
>>> li[1] = 45
>>> li['abc', 45, 4.34, 23]
```

- **We can change lists *in place.***

Tuples: Immutable

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
>>> t[2] = 3.14
Traceback (most recent call last):
File "<pyshell#75>", line 1, in -toplevel-
tu[2] = 3.14
TypeError: object doesn't support item assignment
```

You can’t change a tuple.

Dictionaries

- **Dictionaries store a *mapping* between a set of keys and a set of values.**
- ***Keys can be any immutable type***
- values can be any type
- values and keys can be different types in a single dictionary
- You can define, modify, view, delete

the key-value pairs in the dictionary

Creating and accessing dictionaries

```
>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user']
'bozo'
>>> d['pswd']
1234
>>> d['bozo']
Traceback (innermost last):
File '<interactive input>' line 1, in ?
KeyError: bozo
```

```
>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user'] = 'clown'
>>> d
{'user':'clown', 'pswd':1234}
```

- Keys must be unique
- Assigning to an existing key replaces its value

```
>>> d['id'] = 45
>>> d
{'user':'clown', 'id':45, 'pswd':1234}
```

Removing dictionary entries

```
>>> d = {'user':'bozo', 'p':1234, 'i':34}
>>> del d['user']      # Remove one. Note that del is # a function.
>>> d
{'p':1234, 'i':34}
>>> d.clear()          # Remove all.
>>>d
{}
>>> a=[1,2]
>>> del  a[1]          # (del also works on lists)
>>> a
[1]
```

Useful Accessor Methods

```
>>> d = {'user':'bozo', 'p':1234, 'i':34}

>>> d.keys() # List of currentkeys
['user', 'p', 'i']

>>> d.values() # List of currentvalues.
['bozo', 1234, 34]

>>> d.items() # List of item tuples.
[('user','bozo'), ('p',1234), ('i',34)]
```

Defining Functions

.

```
def get_final_answer(filename):
                    """Documentation String"""
                    line1
                    line2
                    Return total_counter
                    ...
```

Function definition begins with def
Function name and its arguments.
‘return’ indicates the value to be sent back to the caller.

No declaration of types of arguments or result

Calling a Function

```
>>>   def myfun(x, y):
          return x * y
>>>   myfun(3, 4)
12
```

Keyword Arguments

- **Functions can be called with arguments out of order.**
- **These arguments are specified in the call.**
- **Keyword arguments can be used for a final subset of the arguments.**

```
>>> def myfun(a, b, c):
        return a-b
>>> myfun(2, 1, 43)
  1
>> >myfun(c=43, b=1, a=2)
  1
>>> myfun(2, c=43, b=1)
  1
```

Thank you