Spring 2020 SIT22013

ICT Problem Solving

Review on Python Programming

March 4, 2020

Review on Python Programming

- Built-in data types
- Sequence types
- String manipulation
- Functions
- Sorting lists
- Classes
- Useful coding skills

Built-in data types

Data Type	Description	Example
int	integer number	-5, 0, 10, 155
float	floating point number	1.0, 0.05, 3.1415, -11.1
str	string	'Handong', 'John'
bool	logical value	True, False
list	a set of ordered elements	[1, 2, 'John', 4]
tuple	similar to the list but immutable	(1, True, 3, 4)
set	a collection of unordered elements	{1, 2, 3}
dict	a set of unordered key-value pairs	{'a':1, 'b':2, 'c':3}

Sequence types

- data types to represent an ordered set
  e.g., list, tuple, string, etc.

Operation	Result	Example
x in s	True if an item of *s* is equal to *x*, else False	‘b’ in ‘abc’ → True
x not in s	False if an item of *s* is equal to *x*, else True	5 not in [2, 3, 7] → True
s + t	the concatenation of *s* and *t*	(1,2,3) + (4,5,6) → (1, 2, 3, 4, 5, 6)
s * n (or n * s)	equivalent to adding *s* to itself *n* times	‘a’ * 3 → ‘aaa’

Operation	Result	Example
s[i]	*i*-th item of *s*	s = [1,2,3] s[2] → 3
s[i:j]	slice of *s* from *i* to *j*	s = (1,2,3,4,5) s[1:3] → (2,3)
s[i:j:k]	slice of *s* from *i* to *j* with step *k*	s = 'abcdefg' s[1:5:2] → 'bd'
len(s)	length of *s*	s = [1, 5, 'abc'] len(s) → 3
min(s)	smallest item of *s*	s = (5, 14, 3, 8, 9) min(s) → 3
max(s)	largest item of *s*	s = 'python' max(s) → 'y'
s.index(x[,i[, j]])	index of the first occurrence of *x* in *s* (after index *i* and before index *j*)	s = [1,3,5,7,9] s.index(7) → 3
s.count(x)	total number of occurrences of *x* in *s*	s = (1,6,3,1,'ab') s.count(1) → 2

String manipulation

- Editing / Substitution

```
>>> s = ' Hello, World! '
>>> s.strip()
'Hello, World!'
>>> s.replace('World', 'Handong')
' Hello, Handong! '
```

← *Remove white space on the both sides.*

← *Replace a substring by another.*

- Divide / Merge

```
>>> s = 'Hello, how are you?'
>>> s.split()
['Hello,', 'how', 'are', 'you?']
>>>
>>> t = s.split()
>>> ':'.join(t)
'Hello,:how:are:you?'
```

← *Return a list of substrings separated by white space.*

← *Join substrings together to one string by inserting a separator string between consecutive substrings.*

Functions

- What is a function?
  - A callable unit composed of a set of statements
  - Executed through a simple function call by other statements
  - Useful to divide your code into functional units
  - Code reusability and readability can be improved by utilizing functions.

  - **Syntax**

```
def function_name(parameters):
    statements
```

Function without any return value

```
def function_name(parameters):
    statements
    return values
```

Function with return values

- Examples

```
def print_name(name):
    print('My name is',name)
```

```
>>> print_name('Paul')
My name is Paul
```

```
def avg3(x,y,z):
    return (x+y+z)/3
```

```
>>> x = 10
>>> y = 20
>>> z = 30
>>> print('avg3({},{},{}) = {}'.format(x,y,z,avg3(x,y,z)))
avg3(10,20,30) = 20.0
```

- Python allows a function to return multiple values:

```
def div(x,y):
    quotient = x//y
    remainder = x%y
    return quotient, remainder
```

```
>>> a,b = div(10,3)
>>> print(a,b)
3 1
```

Note1: the data type for multiple return values is the tuple.

```
>>> type(div(10,3))
<class 'tuple'>
```

Note2: items in an iterable can be assigned to individual variables:

```
>>> a,b = [2,3]
>>> print(a,b)
2 3
```

```
>>> a,b = (2,3)
>>> print(a,b)
2 3
```

```
>>> a,b = 'cd'
>>> print(a,b)
c d
```

Sorting lists

The **sorted(iterable)** function returns the sorted result of an iterable but the iterable itself does not change.

```
>>> a = [8, 2, 1, 6]
>>> b = a.sort()
>>> print(a)
[1, 2, 6, 8]
>>> print(b)
None
```

Sorting by **sort()** method

```
>>> a = [8, 2, 1, 6]
>>> b = sorted(a)
>>> print(a)
[8, 2, 1, 6]
>>> print(b)
[1, 2, 6, 8]
```

Sorting by **sorted()** function

```
>>> a = ['BB', 'ccc', 'a', 'dddd']
>>> for x in sorted(a, key=str.lower, reverse=True):
        print(x, end=' ')


dddd ccc BB a
```

The key and reverse parameters can still be used in the **sorted()** function.

The **reversed(iterable)** function returns a new iterable to access items in an input iterable in a reverse order but the input iterable itself does not change.

```
>>> a = list(range(5))
>>> b = a.reverse()
>>> print(a)
[4, 3, 2, 1, 0]
>>> print(b)
None
```

Results by **reverse()** method

```
>>> a = list(range(5))
>>> b = reversed(a)
>>> print(a)
[0, 1, 2, 3, 4]
>>> print(b)
<list_reverseiterator object at 0x03479710>
>>> print(list(b))
[4, 3, 2, 1, 0]
```

Results by **reversed()** function

The **reversed()** function is useful when it is used together with a for statement.

```
>>> a = ['Magenta', 'Yellow', 'Green']
>>> a.reverse()
>>> for x in a:
        print(x, end=' ')


Green Yellow Magenta
```

A for statement using **reverse()** method

```
>>> a = ['Magenta', 'Yellow', 'Green']
>>> for x in reversed(a):
        print(x, end=' ')


Green Yellow Magenta
```

A for statement using **reversed()** function

Classes

- The class statement is used to define a class.

```
class <class name>:
    <member variable 1> = <value 1>
    <member variable 2> = <value 2>
    ……
    def <method 1>(self, other arguments):
        <statement block 1>
    def <method 2>(self, other arguments):
        <statement block 2>
    ……
```

The assignment statements add the variables on the left-hand side to the class as member variables.

The method definition must include “self” in the argument list, which represents a class instance.

- Note 2:
  An assignment statement to ***self.<new member>*** in a method adds ***<new member>*** to the class automatically when the statement is executed:

```
class Value:
    def set_val(self,val):
            self.val = val
```

This will add “val” to the “Value” class as its new member when this statement is executed.

At this moment, we have not executed the assignment statement *self.val = val*, and thus “val” member is not available.

```
>>> a = Value()
>>> print(a.val)
Traceback (most recent call last):
  File "<pyshell#297>", line 1, in <module>
    print(a.val)
AttributeError: 'Value' object has no attribute 'val'
>>> a.set_val(10)
>>> print(a.val)
10
```

After calling the set_val function, we are now able to access the “val” member without an exception.

- **Constructor**
  - A method, named “__init__”, which is called for initialization when an instance is generated

- **Destructor**
  - A method, named “__del__”, which is called for finalization when an instance is removed from memory

```
class <class name>:
    ……
    def __init__(self, other arguments):
        <statement block 1>
    def __del__(self):
        <statement block 2>
    ……
```

You may specify additional arguments in the constructor argument list, which can be used for initialization.

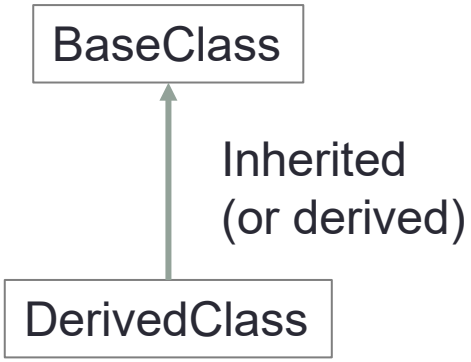
- Example

```
class Person:
    def __init__(self, name):
        self.name = name

    def say_hello(self):
        print('Hello! My name is {}.'.format(self.name))

    def __del__(self):
        print('Instance {} is being removed.'.format(self.name))
```

```
>>> p1 = Person('James')
>>> p1.say_hello()
Hello! My name is James.
>>> del p1
Instance James is being removed.
>>> p1.say_hello()
Traceback (most recent call last):
  File "<pyshell#258>", line 1, in <module>
    p1.say_hello()
NameError: name 'p1' is not defined
```

- In Python, the class inheritance can be done simply specifying a base class name enclosed by parentheses right after its derived class name as follows:

```
class DerivedClass(BaseClass):
    <member definitions>
    <method definitions>
    … …
```




```
class Point1D:
    def __init__(self,x):
        self.x = x
    def print_x(self):
        print('x:',self.x)

class Point2D(Point1D):
    def __init__(self,x,y):
        Point1D.__init__(self,x)
        self.y = y
    def print_y(self):
        print('y:',self.y)
```

```
>>> p = Point2D(10,20)
>>> p.print_x()
x: 10
>>> p.print_y()
y: 20
```

Useful coding skills

- List comprehension
- Lambda functions
- Unpacking

List comprehension

- Provides an easy way to make a list with a few lines

```
>>> s = [k*k for k in range(10)]
>>> print(s)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
s = []
for k in range(10):
    s.append(k*k)
print(s)
```

```
>>> [k*k for k in range(10) if k%2 == 1]
[1, 9, 25, 49, 81]
```

```
s = []
for k in range(10):
    if k%2 == 1:
        s.append(k*k)
print(s)
```

Lambda functions

- What is a lambda function?
  - A function defined in a single line without its name

- Syntax

lambda <parameters> : <expression to return>

```
>>> func = lambda:1
>>> func()
1

>>> func = lambda x,y:x+y
>>> func(10, 20)
30
```

Unpacking

- Assign each item in an iterable on the right-hand side to the corresponding variable on the left-hand side.

```
>>> a,b,c = (1,2,3)
>>> print(a,b,c,sep=',')
1,2,3
```

```
>>> t = tuple(range(5))
>>> a,*b = t
>>> print(a,b,sep=',')
0,[1, 2, 3, 4]
```

```
>>> t = tuple(range(5))
>>> a,b,*c = t
>>> print(a,b,c,sep=',')
0,1,[2, 3, 4]
```

Extended unpacking

- More examples on value assignment with asterisk (*)
  - Passing all the items in an iterable as function arguments.

```
>>> def sum3(a,b,c):
        return a+b+c

>>> t = (1,3,5)
>>> sum3(*t)
9
```

  - Receiving multiple items through a variable-length argument list.

```
>>> def scaled_sum(scale,*items):
        return scale * sum(items)

>>> scaled_sum(0.5,1,2,3,4,5)
7.5
```