Python Advanced

1. Regular expression

A regular expression is a sequence of characters that define a search pattern.

In [1]:
```
# import re module
import re
```

In [2]:
```
print(re.match('www', 'www.huawei.com').span())  # match from start of the text
print(re.match('com', 'www.huawei.com'))        # cannot match from middel of the
```

```
(0, 3)
None
```

In [3]:
```
print(re.search('com', 'www.huawei.com').span())
```

```
(11, 14)
```

In [4]:
```
line = "Cats are smarter than fish"
searchObj = re.search( r'(.*) are smarter than (.*)', line)
if searchObj:
    print("searchObj.group() : ", searchObj.group())
    print("searchObj.group(1) : ", searchObj.group(1))
    print("searchObj.group(2) : ", searchObj.group(2))
else:
    print("Nothing found!!" )
```

```
searchObj.group() :  Cats are smarter than fish
searchObj.group(1) :  Cats
searchObj.group(2) :  fish
```

In [5]:
```
# re.compile(pattern).search(text) is equivalent to re.search(pattern, text)
pattern = re.compile(r'\d+')                    # match at least one digit
n = pattern.match('one12twothree34four')        # match from start. no match
print(n)

# search from start, found 12
m = pattern.search('one12twothree34four')
print(m)
print(m.group())
```

```
None
<re.Match object; span=(3, 5), match='12'>
12
```

In [6]:
```
phone = "2020-0101-000 # this is a phone number"
# remove the number sign (#) and everything behine it
num = re.sub(r'#.*', "", phone)
print("phone number: ", num)
# remove everything that is not digit
num = re.sub(r'\D', "", phone)
print("phone number : ", num)
```

```
phone number:  2020-0101-000
phone number :  20200101000
```

In [7]:
```
# find all the numbers in the text
text = "Tomorrow is 2022/2/31, today is 2022/2/30"
num1 = re.findall(r'\d+', text)
num2 = re.findall(r'[0-9]{2,5}', text) # different regex can lead to same result
print(num1)
print(num2)
```

```
['2022', '2', '31', '2022', '2', '30']
['2022', '31', '2022', '30']
```

In [8]:
```
# find all the alphabets in the text
s = re.findall(r'[a-zA-z]+', text)
print(s)
```

```
['Tomorrow', 'is', 'today', 'is']
```

In [9]:
```
# find all the symbols in the text
s = re.findall(r'\W+', text)
print(s)
```

```
[' ', ' ', '/', '/', ', ', ' ', ' ', '/', '/']
```

In [10]:
```
# find all the alphabets and digits
s = re.findall(r'[A-Za-z0-9]+', text)
print(s)
```

```
['Tomorrow', 'is', '2022', '2', '31', 'today', 'is', '2022', '2', '30']
```

In [11]:
```
# find email address
text = "my email address is: abc456@def.com"
s = re.findall(r'[A-Za-z0-9]+@[A-Za-z0-9]+\.com', text)
print(s)
```

```
['abc456@def.com']
```

In [12]:
```
# find url
text = "Python home page: https://www.python.org"
s = re.findall(r'https?://.*', text)
print(s)
```

```
['https://www.python.org']
```

In [13]:
```
# find every div tag
html = "aa<div>test1</div>bb<div>test2</div>cc "
res = re.search("<div>.*</div>",html)
print(res.group())
```

```
<div>test1</div>bb<div>test2</div>
```

In [14]:
```
# find first div tag
html = "aa<div>test1</div>bb<div>test2</div>cc "
res = re.search("<div>.*?</div>",html)
print(res.group())
```

```
<div>test1</div>
```

2. File I/O

2.1 Python allows you to read, write and delete files

to read a file, we can use:

- open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)
  - -file: a path-like object giving the pathname (absolute or relative to the current working directory) of the file to be opened
  - -mode: describes the way in which the file will be used (r: read only, w: write only, a: append, r+: read and write; *b: opens the file in binary mode)
  - -encoding: specify encoding

example: f = open(“file_name”,“r”,encoding=“utf8”)

In [15]:
```
f = open("text.txt", 'w', encoding='utf8') # open text.txt. if not exist, Python
```

In [16]:
```
inputs = input("input: ")
f.write(inputs)
f.close()
```

```
input: I love Python, Python is cool
```

In [17]:
```
f = open("text.txt", 'r')
print(f.read(6)) # read 6 character from the current position(default is 0).
print(f.read()) # read from current posistion to the end
f.close()
```

```
I love
 Python, Python is cool
```

In [18]:
```
f = open("text.txt", 'a')
f.write(" I appended more content!")
f.close()
```

In [19]:
```
f = open("text.txt", 'r')
print(f.read())
f.close()
```

```
I love Python, Python is cool I appended more content!
```

In [20]:
```
# use with statement to open file
with open("text1.txt", 'w') as f:
    f.write("python is cool\nthis is a pen\nI like apple")
# use with statement to read file
with open("text1.txt", 'r') as f:
    print(f.read())
```

```
python is cool
this is a pen
I like apple
```

In [21]:
```
# open file
with open("text1.txt", "r") as f:
    line = f.readline()
    print ("read one line: %s" % (line))

    lines = f.readlines()
    print(lines)
```

```
read one line: python is cool

['this is a pen\n', 'I like apple']
```

3. Errors and Exceptions

In [22]:
```
this is invalid syntax
```

```
  File "<ipython-input-22-0fbbd1037ae5>", line 1
    this is invalid syntax
                          ^
SyntaxError: invalid syntax
```

In [23]:
```
# Python can't compute 1/0
print(1/0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-23-fc3cfb21a813> in <module>
      1 # Python can't compute 1/0
----> 2 print(1/0)

ZeroDivisionError: division by zero
```

In [24]:
```
# Catch errors or exceptions with try-except statement
try:
    print(1/0)
except Exception as e:
    print(e)
finally:
    print("Python is cool")
```

```
division by zero
Python is cool
```

In [25]:
```
# Catch errors or exceptions with try-except statement
a = input()
try:
    b = [i for i in range(int(a))] # if a<5, there will be an error
    print(b[4]/0)
except ZeroDivisionError:
    print("division by zero")
except ValueError:
    print("ValueError")
except IndexError:
    print("list index out of range")
finally:
    print("Python is cool")
```

```
aaa
ValueError
Python is cool
```

In [26]:
```
# custom exception
class MyError(Exception):
    def __init__(self,ErrorInfo):
        super().__init__(self) #initialize parent class
        self.errorinfo=ErrorInfo
    def __str__(self):
        return self.errorinfo
```

In [27]:
```
# throw an exception
raise MyError("my exception")
```

```
---------------------------------------------------------------------------
MyError                                   Traceback (most recent call last)
<ipython-input-27-75cc95ad5cb3> in <module>
      1 # throw an exception
----> 2 raise MyError("my exception")

MyError: my exception
```

In [28]:
```
# assert statement
def func(a,b):
    # if not a==b, Python will throw an exception
    assert a==b
func(1,2)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-28-73be1703f7f8> in <module>
      3     # if not a==b, Python will throw an exception
      4     assert a==b
----> 5 func(1,2)

<ipython-input-28-73be1703f7f8> in func(a, b)
      2 def func(a,b):
      3     # if not a==b, Python will throw an exception
----> 4     assert a==b
      5 func(1,2)

AssertionError:
```

4. Generator and decorator

4.1 Generator

In [30]:
```
# use isinstance function to check if an object is iterable
import collections as c
print(isinstance([], c.Iterable))
print(isinstance('abc', c.Iterable))
print(isinstance(100, c.Iterable))
```

```
True
True
False

C:\Users\lWX992962\Anaconda3\envs\python_course\lib\site-packages\ipykernel_lau
ncher.py:3: DeprecationWarning: Using or importing the ABCs from 'collections'
instead of from 'collections.abc' is deprecated since Python 3.3,and in 3.9 it
will stop working
  This is separate from the ipykernel package so we can avoid doing imports unt
il
```

In [31]:
```
l = [1, 2, 3, 4, 5]
l_iter = iter(l)
print(type(l_iter))
```

```
<class 'list_iterator'>
```

In [32]:
```
next(l_iter)
```

Out[32]:
```
1
```

In [33]:
```
next(l_iter)
```

Out[33]:
```
2
```

In [34]:
```
# ues yeild statement to construct a generator for fibonacci sequence
def fib(n):
    current = 0
    num1, num2 = 0, 1
    while current < n:
        num = num1
        num1, num2 = num2, num1+num2
        current += 1
        yield num
    yield "done"

g=fib(5)

for x in g:
    print(x)
```

```
0
1
1
2
3
done
```

In [35]:
```
g = ( x*2 for x in range(5))

for x in g:
    print(x)
```

```
0
2
4
6
8
```

4.2 Decorator

In [36]:
```
def decorate(func):
    def decorated():
        print("I got decorated")
        func()
    return decorated

@decorate
def plain():
    print("I am plain")

plain()
```

```
I got decorated
I am plain
```

5. Python Standard Library

There are two types of libraries in Python, the standard library and third-party libraries

- Standard library: comes with Python. For example, os, sys and time.
- Third-party libraries: needs to be installed be for it can be used. For example, numpy, pandas and scikit-learn

5.1 sys module

provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.

In [37]:
```
# sys.exit([n]): can be used to exit the program, n=0 means successful terminatic
import sys
for i in range(100):
    print(i)
    if i == 5:
        sys.exit(0)
```

```
0
1
2
3
4
5

An exception has occurred, use %tb to see the full traceback.

SystemExit: 0



C:\Users\lWX992962\Anaconda3\envs\python_course\lib\site-packages\IPython\core
\interactiveshell.py:3426: UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

In [38]:
```
# sys.path: a list of strings that specifies the search path for modules.
sys.path
```

Out[38]:
```
['D:\\lab\\code',
 'C:\\Users\\lWX992962\\Anaconda3\\envs\\python_course\\python37.zip',
 'C:\\Users\\lWX992962\\Anaconda3\\envs\\python_course\\DLLs',
 'C:\\Users\\lWX992962\\Anaconda3\\envs\\python_course\\lib',
 'C:\\Users\\lWX992962\\Anaconda3\\envs\\python_course',
 '',
 'C:\\Users\\lWX992962\\Anaconda3\\envs\\python_course\\lib\\site-packages',
 'C:\\Users\\lWX992962\\Anaconda3\\envs\\python_course\\lib\\site-packages\\win
32',
 'C:\\Users\\lWX992962\\Anaconda3\\envs\\python_course\\lib\\site-packages\\win
32\\lib',
 'C:\\Users\\lWX992962\\Anaconda3\\envs\\python_course\\lib\\site-packages\\Pyt
honwin',
 'C:\\Users\\lWX992962\\Anaconda3\\envs\\python_course\\lib\\site-packages\\IPy
thon\\extensions',
 'C:\\Users\\lWX992962\\.ipython']
```

In [39]:
```
# sys.platform: used to check the system platform.
sys.platform
```

Out[39]:
```
'win32'
```

5.2 os module

provides a portable way of using operating system dependent functionality.

In [40]:
```
import os
# os.getpid() get the current process id
print("process id: ", os.getpid())

# os.getppid(): get the process ID (PID) of the calling process.
print("parent process id：", os.getppid())

# os.getcwd(): get the current directory
cwd = os.getcwd()
print("current directory: ", cwd)

# os.chdir(path): change current directory
os.chdir("C:/")
print("new directory: ", os.getcwd())

# os.listdir(path): get a list containing the names of the entries in the directo
print("all the files: ", os.listdir(cwd))
```

```
process id:  5336
parent process id：  18016
current directory:  D:\lab\code
new directory:  C:\
all the files:  ['.ipynb_checkpoints', '02 Python Basic.ipynb', '03 Python adva
nced-Copy1.ipynb', '03 Python advanced.ipynb', '04 Third-Party libraries.ipyn
b', '05 Python exercise.ipynb', 'covid19.csv', 'text.txt', 'text1.txt', 'xxx.PN
G']
```

In [41]:
```
# since we've changed the current directory, remember to change it back now
os.chdir("D:/lab/code") # change the path to your notebook location
```

In [42]:
```
# os.walk(): print all the file names under the current directory (including file
for root, dirs, files in os.walk(cwd):
    for name in files:
        print(os.path.join(root, name))
    for name in dirs:
        print(os.path.join(root, name))
```

```
D:\lab\code\02 Python Basic.ipynb
D:\lab\code\03 Python advanced-Copy1.ipynb
D:\lab\code\03 Python advanced.ipynb
D:\lab\code\04 Third-Party libraries.ipynb
D:\lab\code\05 Python exercise.ipynb
D:\lab\code\covid19.csv
D:\lab\code\text.txt
D:\lab\code\text1.txt
D:\lab\code\xxx.PNG
D:\lab\code\.ipynb_checkpoints
D:\lab\code\.ipynb_checkpoints\02 Python Basic-checkpoint.ipynb
D:\lab\code\.ipynb_checkpoints\03 Python advanced-checkpoint.ipynb
D:\lab\code\.ipynb_checkpoints\03 Python advanced-Copy1-checkpoint.ipynb
D:\lab\code\.ipynb_checkpoints\04 Third-Party libraries-checkpoint.ipynb
D:\lab\code\.ipynb_checkpoints\05 Python exercise-checkpoint.ipynb
```

In [43]:
```
# os.path module: provides some useful functions on pathnames
import os
# os.path.abspath(path): get the absolute pathname
print("absolute pathname: ",os.path.abspath("text.txt"))

# os.path.exists(path): return ture if path exists, else return false.
print("exists or not: ",os.path.exists("text.txt"))

# os.path.getsize(path): return the size of path
print("size: ",os.path.getsize("text.txt"))

# os.path.isfile(path): determine if path is a file
print("is file: ",os.path.isfile("text.txt"))

# os.path.isdir(path): determine if path is a folder
print("is folder: ",os.path.isdir("text.txt"))
```

```
absolute pathname:  D:\lab\code\text.txt
exists or not:  True
size:  54
is file:  True
is folder:  False
```

5.3 time module

provides various time-related functions.

In [44]:
```
import time
# time.time(): get current timestamp
time_now = time.time()
print("time stamp: ",time_now)
```

```
time stamp:  1609410362.9803462
```

In [45]:
```
# time.localtime(): get local time
localtime = time.localtime(time_now)
print("local time: ", localtime)
```

```
local time:  time.struct_time(tm_year=2020, tm_mon=12, tm_mday=31, tm_hour=18,
tm_min=26, tm_sec=2, tm_wday=3, tm_yday=366, tm_isdst=0)
```

In [46]:
```
# time.asctime(): convert a tuple or struct_time representing a time to a string
localtime = time.asctime(localtime)
print("local time: ", localtime)
```

```
local time:  Thu Dec 31 18:26:02 2020
```

In [47]:
```
#time.strftime(format[, t]): Convert a tuple or struct_time representing a time t
print("local time: ", time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))
```

```
local time:  2020-12-31 18:26:04
```

6. Data Structure

6.1 Singly linked list

6.1.1 Define the basic element, node, of the linked list

In [48]:
```
# define the basic element that forms the list.
class Node:
    # constructor
    def __init__(self, data=None, next_node=None):
        self.__data = data
        self.__next_node = next_node
    # get data
    def get_data(self):
        return self.__data
    # get next node
    def get_next(self):
        return self.__next_node
    # set next node
    def set_next(self, new_next):
        self.__next_node = new_next
```

6.1.2 Define the list

In [49]:

```
class SinglyLinkedList:

    # constructor
    def __init__(self):
        self.__head = Node('__head__')

    # get the first node that contains the specified data
    def get_node(self, data):
        current = self.__head

        # go through the list until it finds a match, or reach the end of the lis
        while current:
            if current.get_data() == data:
                return current
            else:
                current = current.get_next()

        return None

    # delete the first node that contains the specified data
    def delete(self, data):
        current = self.__head
        previous = None
        # go through the list until it finds a match, or reach the end of the lis
        while current:
            if current.get_data() == data:
                previous.set_next(current.get_next())
                break;
            else:
                previous = current
                current = current.get_next()



    # append new node to the end of the list
    def append(self, data):
        current = self.__head
        # go to the last node in the list
        while current.get_next():
            current = current.get_next()
        # append at the end of the list
        current.set_next(Node(data))

    # get the number of nodes in the list
    def size(self):
        current = self.__head
        count = 0
        while current:
            count += 1
            current = current.get_next()
        return count-1

    def print_list(self):
        current = self.__head.get_next()
        while current:
            print(current.get_data())
```

```
            current = current.get_next()
```

6.1.3 Test our list

In [50]:
```
# create list
l = SinglyLinkedList()
```

In [51]:
```
l.append('cat')
l.print_list()
```

```
cat
```

In [52]:
```
# append
l.append('dog')
l.append('fish')
l.append('bird')
l.print_list()
```

```
cat
dog
fish
bird
```

In [53]:
```
# get_node
node = l.get_node('fish')
print(node.get_data())
```

```
fish
```

In [54]:
```
# delete
l.delete('fish')
l.print_list()
```

```
cat
dog
bird
```

In [55]:
```
# size
l.size()
```

Out[55]:
```
3
```

6.2 Doubly linked list

6.2.1 Define the basic element, node, of the linked list

In [56]:
```python
# define the basic element that forms the list.
class Node:
    # constructor
    def __init__(self, data=None, next_node=None, prev_node=None):
        self.__data = data
        self.__next_node = next_node
        self.__prev_node = prev_node
    # get data
    def get_data(self):
        return self.__data
    # get next node
    def get_next(self):
        return self.__next_node
    # set next node
    def set_next(self, new_next):
        self.__next_node = new_next


    # get prev node
    def get_prev(self):
        return self.__prev_node
    # set prev node
    def set_prev(self, new_prev):
        self.__prev_node = new_prev
```

6.2.2 Define the list

In [57]:

```
class DoublyLinkedList:

    # constructor
    def __init__(self):
        head = Node('__head__')
        self.__head = head
        self.__tail = head

    # get the first node that contains the specified data
    def get_node(self, data):
        current = self.__head

        # go through the list until it finds a match, or reach the end of the lis
        while current:
            if current.get_data() == data:
                return current
            else:
                current = current.get_next()

        return None

    # append new node to the end of the list
    def append(self, data):
        new_tail = Node(data)
        self.__tail.set_next(new_tail)
        new_tail.set_prev(self.__tail)
        self.__tail = new_tail

    # delete the first node that contains the specified data
    def delete(self, data):
        del_node = self.get_node(data)
        if del_node:
            prev_node = del_node.get_prev()
            next_node = del_node.get_next()
            prev_node.set_next(next_node)
            if next_node:
                next_node.set_prev(prev_node)
            else:
                self.__tail = prev_node

    # get the number of nodes in the list
    def size(self):
        current = self.__head
        count = 0
        while current:
            count += 1
            current = current.get_next()
        return count-1

    def print_list(self):
        current = self.__head.get_next()
        while current:
            print(current.get_data())
            current = current.get_next()

    def print_backwards(self):
```

```
        current = self.__tail
        while current.get_prev():
            print(current.get_data())
            current = current.get_prev()
```

6.2.3 Test our list

In [58]:
```
# create list
l = DoublyLinkedList()
```

In [59]:
```
# append
l.append('cat')
l.append('dog')
l.append('fish')
l.append('bird')
l.print_list()
```

```
cat
dog
fish
bird
```

In [60]:
```
l.print_backwards()
```

```
bird
fish
dog
cat
```

In [61]:
```
# delete
l.delete('cat')
l.print_list()
```

```
dog
fish
bird
```

In [62]:
```
# size
l.size()
```

Out[62]:
```
3
```

6.3 Binary tree

A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.

Traversal is a process of visiting all the nodes of a tree. There are three ways which we use to traverse a tree:

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

In [63]:
```
# BinaryTreeNode
class BinaryTreeNode(object):
    def __init__(self):
        self.data = '#'
        self.leftChild = None
        self.rightChild = None
```

In [64]:

```
class BinaryTree(object):
    #create binary tree
    def createBinaryTree(self, Root):
        data = input('==>')
        if data == '#':
            Root = None
        else:
            Root.data = data
            Root.leftChild = BinaryTreeNode()
            self.createBinaryTree(Root.leftChild)
            Root.rightChild = BinaryTreeNode()
            self.createBinaryTree(Root.rightChild)


    def preOrder(self, Root):
        if Root is not None:
            self.visitBinaryTreeNode(Root)
            self.preOrder(Root.leftChild)
            self.preOrder(Root.rightChild)


    def inOrder(self, Root):
        if Root is not None:
            self.inOrder(Root.leftChild)
            self.visitBinaryTreeNode(Root)
            self.inOrder(Root.rightChild)


    def postOrder(self, Root):
        if Root is not None:
            self.postOrder(Root.leftChild)
            self.postOrder(Root.rightChild)
            self.visitBinaryTreeNode(Root)


    def visitBinaryTreeNode(self, BinaryTreeNode):
        # pound sign (#) means empty node.
        if BinaryTreeNode.data is not '#':
            print(BinaryTreeNode.data, end="->")
```

In [65]:
```
bTN = BinaryTreeNode()
bT = BinaryTree()
bT.createBinaryTree(bTN)
print('pre_order: ')
bT.preOrder(bTN)
print('\nin_order: ')
bT.inOrder(bTN)
print('\npost_order: ')
bT.postOrder(bTN)
```

```
==>5
==>2
==>#
==>#
==>8
==>7
==>#
==>#
==>10
==>#
==>#
pre_order:
5->2->8->7->10->
in_order:
2->5->7->8->10->
post_order:
2->7->10->8->5->
```

In [ ]: