Lecture 11

Namespaces, Recursion and files

Namespaces

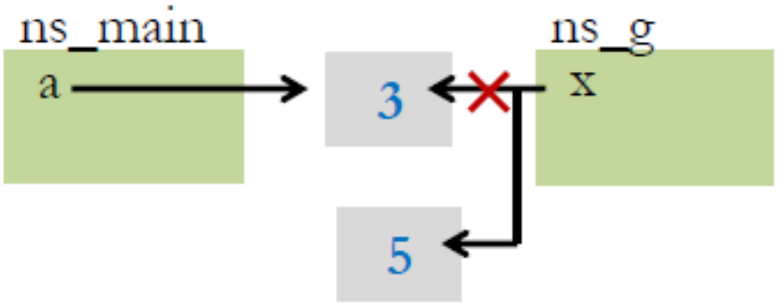
- Namespace is an abstract container or environment created to hold a logical grouping of identifiers / variables.
- An identifier/variable defined in a namespace is usually associated only with that namespace.
- The same identifier/variable can be independently defined in multiple namespaces.
- An identifier/variable defined in one namespace may or may not interfere with the identifier/variable defined in another namespace.
- Languages that support namespaces specify the rules that determine to which namespace an identifier/variable belongs.

Parameter passing in Namespaces

Passing Immutable Objects as Parameters

Example 1: Consider the following code:

```
def g(x):        Step 3
    x=5          Step 4
a=3              Step 1
g(a)             Step 2
print(a)         Step 5
```



Output
3

Problem

Write a function that inputs a list of integers from user.

```
def lst_input():
    my_str=input('Enter integer values separated by commas:')
    my_str=my_str.split(',')
    int_lst=[]
    for i in my_str:
        int_lst.append(int(i))
    return int_lst
print('You entered:',lst_input())
```

To separate the elements from that list and to convert string into integer because this is a string. Split function divide a string into smaller strings depending on the delimiter, delimiter here is ',' is the point at which python will break that string. If you do not pass anything, default is space.

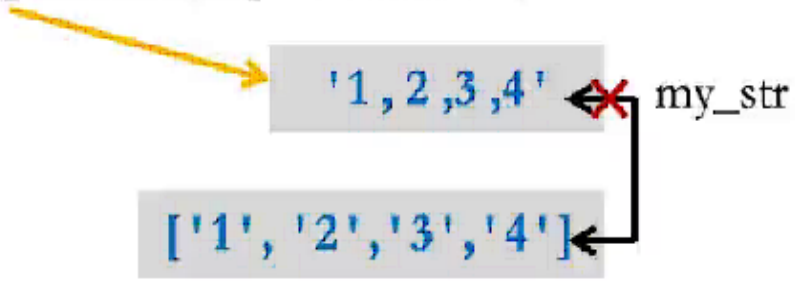
Problem

Write a function that inputs a list of integers from user.

```
def lst_input():
    my_str=input('Enter integer values separated by commas:')
    my_str=my_str.split(',')
    int_lst=[]
    for i in my_str:
        int_lst.append(int(i))
    return int_lst

print('You entered:',lst_input())
```

Execution starts here



Let suppose user enter, '1,2,3,4' … the new link has been created, but this link also contain string, so initializing empty list.

Problem

Write a function that inputs a list of integers from user.

```
def lst_input():
    my_str=input('Enter integer values separated by commas:')
    my_str=my_str.split(',')
    int_lst=[]
    for i in my_str:
        int_lst.append(int(i))
    return int_lst

print('You entered:',lst_input())
```

Execution starts here

'1,2,3,4' ← my_str

['1', '2','3','4']

[ ] ← int_lst

[1,2,3,4] ← int_lst

Output

Enter integer values separated by commas: 1,2,3,4

You entered: [1,2,3,4]

Problem

Write a function that inputs a list of integers from user.

```
def lst_input():
    my_str=input('Enter integer values separated by commas:')
    my_str=my_str.split(',')
    int_lst=[]
    for i in my_str:
        int_lst.append(int(i))
    return int_lst
print('You entered:',lst_input())
```

```
/namespace 1.py
Enter integer values separated by commas:3,5,6,8,9,4
You entered: [3, 5, 6, 8, 9, 4]
>>>
```

Problem

Write a function that inputs a list of integers from user.

Method -2

```
def lst_input(lst):
    my_str=input('Enter integer values separated by commas:')
    my_str=my_str.split(',')
    for i in my_str:
        lst.append(int(i))

L=[]
lst_input(L)
print('You entered:',L)
```

Output

Enter integer values separated by commas: 1,33,65,-1

You entered: [1,33,65,-1]

- What will be the output if last two statements are replaced with: `print(lst_input(L))`

Problem

Write a function that inputs a list of integers from user.
Method -2

```
def lst_input(lst):
    my_str=input('Enter integer values separated by commas:')
    my_str=my_str.split(',')
    for i in my_str:
        lst.append(int(i))

L=[]
lst_input(L)
print('You entered:',L)
```

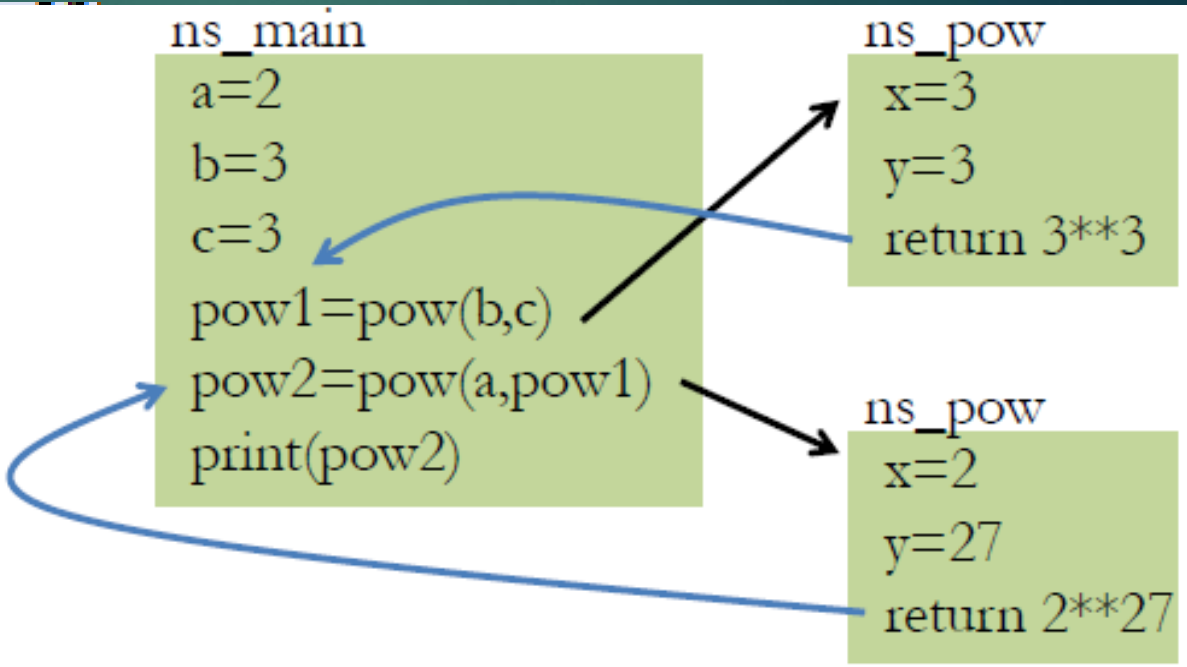
Output

Enter integer values separated by commas: 1,33,65,-1
You entered: [1,33,65,-1]

Trace the Codes

Example 1: Code to find ***abc*** (a**b**c)

```
def pow(x,y):
    return x**y
a=2
b=3
c=3
pow1=pow(b,c)
pow2=pow(a,pow1)
print(pow2)
```



Recursion

Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

Recursion

```
def recur_factorial(n):
    if n == 1:
        return n
    else:
        return n*recur_factorial(n-1)

num = 7

# check if the number is negative
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    print("The factorial of", num, "is", recur_factorial(num))
```

Recursion

A recursive function is a function that calls itself.
The idea is to represent a problem in terms of one or more smaller problems.

Components / properties of a recursive function:

Base Case
- Indicates the stopping condition.
- Could be more than one.

Recursive Call
- Moves the execution towards the base case.
- Could be more than one.

Why Use Recursion??

- Recursion is made for solving problems that can be broken down into smaller, repetitive problems.
- It is especially good for working on things that have many possible branches and are too complex for an iterative approach.
- Recursion may be efficient from programmer's point of view.
- Recursion may not be efficient from computer's point of view.

Files

Files

A file is a sequence of bytes stored on a secondary memory device.

Files are of various types:

- Text Files
- Spreadsheets
- Binary Files
- Executable Files

Files are managed by File system that operating system supports.

Processing a file is based on three steps:

- Operating a file for reading or writing
- Reading from or writing to the file
- Closing the file

Files

Python Standard library includes a module names ‘io’ which contains class for handling files called ‘TextIoWrapper’.

```
>>> io.TextIOWrapper
<class '_io.TextIOWrapper'>
```

Since filing is a very common activity , no import is required to access these functions.

Opening a file

The function open() is used to open the files (text or binary). This function is defined in built-in modules.

The function takes:

- A file name (with or without path)
- '\' is used for path but since it may coincide with escape sequence so python accepts '/' (forward slash).
- A path could be absolute or relative:

  - Raw / absolute path starts from root directory:

  e.g.: c:\office\classes\CP\Text.txt

  - Relative path starts the sequence from current directory:

  e.g: CP\Text.txt

Opening a file

The function takes:

- A file name (with or without path).
- Mode specifies how to interact with opened file.
- r=>reading mode (default)
- w=>writing mode, if the file already exists otherwise its content is wiped out
- a=>append mode, the data will append to the end of file.
- t=>text mode (default)
- b=>binary mode

Opening a file

>>> f=open('myfile.txt') ≡ f=open('myfile.txt', 'r')

- Opens myfile.txt if it exist ; returns an object of io.TextIOWrapper type simply called File Type.
- Generates error if the file does not exist.
- The file is opened for reading only.

>>> f=open('myfile.txt', 'w')

- Opens myfile.txt for writing.
- Creates a new file if it doesn't exist.
- Overwrites the existing file.

Opening a file

```
>>> f=open('myfile.txt', 'a')
```

- Opens myfile.txt for writing.
- Creates a new file if it doesn't exist.
- Appends at the end of existing file.

with/as statement

- Automatically closes a file after block of code is executed.

Syntax:

```
with open <file name> as f:
    <block>
```

- Opens <file name> and assigns it handler.
- Closes f after <block> is executed.

Reading a file

f.read(n)

- reads and returns as string ‘n’ characters from file
- ‘f’ or until the end of file is reached.

f.read()

- reads and returns as string characters from file f until the end of file.

f.readline()

- reads and returns as string characters from file f until (including) new line character or end of file.

f.readlines()

- reads and returns as list.

Reading a file

- With every opened file, the system will associate a cursor that points to the character in the file.
- When the file is first opened, the cursor typically points to the start of the file.
- Using different types of read operations consecutively, second read commences from where first read ended.

Other useful Functions on files

- f.name

  Contains name of file. It's an attribute, not a method.
- f.seek(offset, from_what)

  Changes file object position.

  Position is computed from adding offset to a reference point

  Reference point is selected by from_what argument.

  - From_what=0, offset measured from start of file.
  - From_what=1, offset measured from current position.
  - From_what=2, offset measured from EoF
  - Default value is 0
- f.tell(): returns an integer giving file objects current position in the file as number of bytes from the beginning of the file.

Example -1

Store the following file as myfile.text

myfile.txt - Notepad

File Edit Format View Help

```
First line
Second line
Third line
Last line
```

```
>>> f=open('myfile.txt')
>>> f.read()
'First line\nSecond line\nThird line\nLast line'
>>> f.read()
''
>>> f.close()
>>> f=open('myfile.txt')
>>> print(f.read())
First line
Second line
Third line
Last line
>>> f.close()
```

Example -2

Reading the files 10 characters at a time

File Edit Format View Help

```
First line
Second line
Third line
Last line
```

```
>>> f=open('myfile.txt')
>>> f.read(10)
'First line'
>>> f.read(10)
'\nSecond li'
>>> f.read(10)
'ne\nThird l'
>>> f.close()
```

Example -3

Reading one line at a time

File Edit Format View Help

```
First line
Second line
Third line
Last line
```

```
>>> f=open('myfile.txt')
>>> f.readline()
'First line\n'
>>> f.readline()
'Second line\n'
>>> f.readline()
'Third line\n'
>>> f.close()
```

Example - 4

Reading all lines

```
>>> f=open('myfile.txt')
>>> f.readlines()
['First line\n', 'Second line\n', 'Third line\n', 'Last line']
>>> f=open('myfile.txt')
>>> p=f.readlines()
>>> print(p[1])
Second line

>>> f.close()
```

Example - 5

```
>>> f=open('myfile.txt')
>>> f.read()
'First line\nSecond line\nThird line\nLast line'
>>> f.read()
''
>>> f.seek(0)
0
>>> f.read()
'First line\nSecond line\nThird line\nLast line'
>>> f.seek(3)
3
>>> f.read(5)
'st li'
>>> f.tell()
8
```

Thank you!