Computing Laboratory

Basics of Python - III

Malay Bhattacharyya

Associate Professor

MIU, CAIML, TIH
Indian Statistical Institute, Kolkata

August, 2023

Standard Input/Output functions

I/O from the terminal:

Value can be printed without mentioning the type as:

```
print(*obj,sep=' ',end='\n',file=sys.stdout,flush=False)
```

Value is taken in a string and can be converted to appropriate type using int(), float(), bool(), etc. as:

```
input([prompt])
```

I/O from files:

- `open()`, `close()` # Files are opened in r/w/a mode and the address is returned to a file pointer
- `read()`, `write()` # With a file pointer
- `readline()` # With a file pointer
- `readlines()`, `writelines()` # With a file pointer

print() – Special features

```
a, b, c = 1, 2, 3
print(a, b, c, sep = '')
print(a, b, c, sep = '-')
print(a, b, c, sep = '1.1')
x = print('Hi')
print(x)
print(type(x))
```

print() – Special features

```
a, b, c = 1, 2, 3
print(a, b, c, sep = '')
print(a, b, c, sep = '-')
print(a, b, c, sep = '1.1')
x = print('Hi')
print(x)
print(type(x))
```

Output:

```
123
1-2-3
11.121.13
Hi
None
<class 'NoneType'>
```

print() – Special features

```
a = 7
print(a, 'is prime', end = ';') # ',' includes a space
b = 'prime'
print('7 is ' + b, end = ';') # '+' works only on strings
```

print() – Special features

```
a = 7
print(a, 'is prime', end = ';') # ',' includes a space
b = 'prime'
print('7 is ' + b, end = ';') # '+' works only on strings
```

Output:

```
7 is prime;7 is prime;
```

print() – Special features

```
ls = [[1, 2, 3], [4, 5, 6]]
for i in range(len(ls)):
    for j in range(len(ls[0])):
        print(ls[i][j], end = ' ')
    print()
```

print() – Special features

```
ls = [[1, 2, 3], [4, 5, 6]]
for i in range(len(ls)):
    for j in range(len(ls[0])):
        print(ls[i][j], end = ' ')
    print()
```

Output:

```
1 2 3
4 5 6
```

print() – Special features

```
inputFile = open('test.txt', 'w')
print('Write your own fate!!!', file = inputFile)
inputFile.close()
```

print() – Special features

```
inputFile = open('test.txt', 'w')
print('Write your own fate!!!', file = inputFile)
inputFile.close()
```

Output:

```
Write your own fate!!!
```

- written within test.txt.

input() – Special features

```
x = int(input())
print(x)
n = input('Enter three integers: ')
print(n, list(n))
n1, n2, n3 = input('Enter three integers: ').split()
print(n1+n2+n3, int(n1)+int(n2)+int(n3))
```

input() – Special features

```
x = int(input())
print(x)
n = input('Enter three integers: ')
print(n, list(n))
n1, n2, n3 = input('Enter three integers: ').split()
print(n1+n2+n3, int(n1)+int(n2)+int(n3))
```

Output:

```
10
10
Enter three integers: 1 2 3
1 2 3 ['1', ' ', '2', ' ', '3']
Enter three integers: 1 2 3
123 6
```

input() – Special features

```
r = int(input('Enter the number of rows: '))
c = int(input('Enter the number of columns: '))
MAT = [[int(input()) for i in range(c)] for j in range(r)]
print(MAT)
```

input() – Special features

```
r = int(input('Enter the number of rows: '))
c = int(input('Enter the number of columns: '))
MAT = [[int(input()) for i in range(c)] for j in range(r)]
print(MAT)
```

Output:

```
Enter the number of rows: 2
Enter the number of columns: 3
1
2
3
4
5
6
[[1, 2, 3], [4, 5, 6]]
```

`input()` – Special features

Explore the `reshape()` function!!!

Reading data from file

```
def read(file):
    f = open(file, 'r')
    output = f.read()
    f.close()
    return output
output = read('Data.txt')
```

Reading data from file

```
def read(file):
    f = open(file, 'r')
    output = f.read()
    f.close()
    return output
output = read('Data.txt')
```

Reading data from file (alternative approach):

```
with open('Data.txt', 'r') as f: output = f.read();
```

Special Input/Output functions

Reading data from a CSV file:

```
import pandas as pd # Import pandas
pd.read_csv("file.csv") # reading CSV file
```

Special Input/Output functions

Reading data from a CSV file:

```
import pandas as pd # Import pandas
pd.read_csv("file.csv") # reading CSV file
```

Reading data from an XLS file:

```
import pandas as pd # Import pandas
pd.read_excel("file.xls") # supports old XLS file formats
pd.read_excel("file.xls", engine='openpyxl') # new formats
```

Special Input/Output functions

Reading data from a CSV file:

```
import pandas as pd # Import pandas
pd.read_csv("file.csv") # reading CSV file
```

Reading data from an XLS file:

```
import pandas as pd # Import pandas
pd.read_excel("file.xls") # supports old XLS file formats
pd.read_excel("file.xls", engine='openpyxl') # new formats
```

Note: The Python library `openpyxl` must be used to read/write Excel 2010 xlsx files.

Functions

```
def <function-name>(<argument 1>, ..., <argument n>):
    Statement 1
    Statement 2
    Statement 3
    return <expression>
```

Functions

```
def <function-name>(<argument 1>, ..., <argument n>):
    Statement 1
    Statement 2
    Statement 3
    return <expression>
```

Note: You may either return a values obtained from an `<expression>` or return nothing based on your requirement.

Functions – Understanding the internals

```
def function(a, b, c):
    print('Inside 1:', hex(id(a)), hex(id(b)), hex(id(c)))
    b += 10
    print('Inside 2:', hex(id(a)), hex(id(b)), hex(id(c)))
x, y, z = 10, 10, 20
print('Outside:', hex(id(x)), hex(id(y)), hex(id(z)))
function(x, y, z)
```

Functions – Understanding the internals

```
def function(a, b, c):
    print('Inside 1:', hex(id(a)), hex(id(b)), hex(id(c)))
    b += 10
    print('Inside 2:', hex(id(a)), hex(id(b)), hex(id(c)))
x, y, z = 10, 10, 20
print('Outside:', hex(id(x)), hex(id(y)), hex(id(z)))
function(x, y, z)
```

Output:

```
Outside: 0x7fc4436a0210 0x7fc4436a0210 0x7fc4436a0350
Inside 1: 0x7fc4436a0210 0x7fc4436a0210 0x7fc4436a0350
Inside 2: 0x7fc4436a0210 0x7fc4436a0350 0x7fc4436a0350
```

Comparing the lengths of two integers

```
def compare(m, n):
    while m and n:
        m //= 10
        n //= 10
    return m-n
x, y = input('Enter two numbers: ').split()
f = compare(int(x), int(y))
if f > 0:
    print(x, 'has larger number of digits')
elif f < 0:
    print(y, 'has larger number of digits')
else:
    print('Both have same number of digits')
```

Finding prefixes of a string

```
def prefix(str):
    start, end = 0, 0
    while start < len(str):
        if str[end] == str[end-start]:
            print(str[start:end + 1], end= " ")
            end += 1
            if end == len(str):
                start += 1
                end = start
        else:
            start += 1
            end = start # Index of substring
```

Input: prefix('Python')

Lambda functions

Python provides an anonymous function `lambda` that can take any number of arguments, but can only have one expression.

```
x = lambda a: a + 10
print(x(5))
```

Here, the output will be 15.

Lambda functions

```
a, b = 3, 5
maximum = lambda a, b: a if a > b else b
print(f'{maximum(a,b)} is the maximum among', a, 'and', b)
```

Lambda functions

```
a, b = 3, 5
maximum = lambda a, b: a if a > b else b
print(f'{maximum(a,b)} is the maximum among', a, 'and', b)
```

Output:

```
5 is the maximum among 3 and 5
```

Lambda functions

```
l = lambda n: n
q = lambda n: n ** 2
c = lambda n: n ** 3
x = 2
y = c(x) + 3*q(x) + 2*l(x) + 1 # y = x^3 + 3x^2 + 2x + 1
print(y)
```

Lambda functions

```
l = lambda n: n
q = lambda n: n ** 2
c = lambda n: n ** 3
x = 2
y = c(x) + 3*q(x) + 2*l(x) + 1 # y = x^3 + 3x^2 + 2x + 1
print(y)
```

Output:

```
25
```

Late binding closures

Python's closures are late binding. Hence, the values of variables used in closures are looked up when the inner function is called.

```
def increase():
    return [lambda x : i + x for i in range(5)]
for add in increase():
    print(add(10))
```

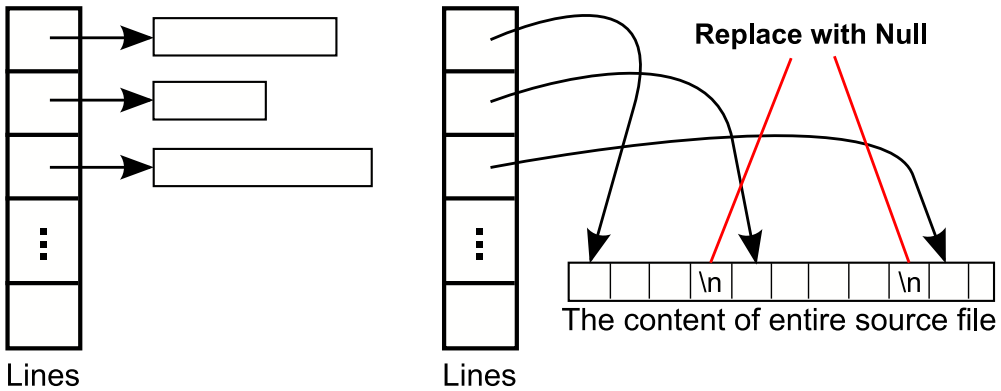
Here, whenever any of the returned functions are called, the value of i is looked up in the surrounding scope at call time. By then, the loop has completed and i is left with its final value of 4. So, it will print {14, 14, 14, 14, 14} instead of {10, 11, 12, 13, 14}.

Problems – Day 4

1. Write a program to reverse the contents of another input file.
2. Suppose a list of strings are given as user input. Write a program to verify whether the ordering of distinct characters in the last string preserves the ordering of characters in all the preceding strings. Note that, conflicts of ordering might exist in the preceding strings.
   Note: The inputs `apple, apollo, apl` preserves the ordering but not the inputs `capacity, pacific, cap`.

Problems – Day 4

3. You have to write a program that reads its own source file (i.e., mtc23xx-day4-prog2.py), and prints the lines in that file in lexicographically sorted order.
   Note: An efficient implementation is highlighted below.



Naive approach Efficient approach