48.what is raw\_input and input?

*raw\_input() takes exactly what the user typed and passes it back as a string. input() takes the raw\_input() and performs an eval() on it as well. The main difference is that input() expects a syntactically correct python statement where raw\_input() does not.51.how to write tables?*

56.how to modify string?

## String Manipulation

To manipulate strings, we can use some of Pythons built-in methods.

Creation

**word = "Hello World"**

**>>> print word**

**Hello World**

Accessing

Use [ ] to access characters in a string

**word = "Hello World"**

**letter=word[0]**

**>>> print letter**

**H**

Length

**word = "Hello World"**

**>>> len(word)**

**11**

Finding

**word = "Hello World"**

**>>> print word.count('l') *# count how many times l is in the string***

**3**

**>>> print word.find("H") *# find the word H in the string***

**0**

**>>> print word.index("World") *# find the letters World in the string***

**6**

Count

**s = "Count, the number of spaces"**

**>>> print s.count(' ')**

**8**

Slicing

Use [ # : # ] to get set of letter

Keep in mind that python, as many other languages, starts to count from 0!!

**word = "Hello World"**

**print word[0] *#get one char of the word***

**print word[0:1] *#get one char of the word (same as above)***

**print word[0:3] *#get the first three char***

**print word[:3] *#get the first three char***

**print word[-3:] *#get the last three char***

**print word[3:] *#get all but the three first char***

**print word[:-3] *#get all but the three last character***

**word = "Hello World"**

**word[start:end] *# items start through end-1***

**word[start:] *# items start through the rest of the list***

**word[:end] *# items from the beginning through end-1***

**word[:] *# a copy of the whole list***

Split Strings

**word = "Hello World"**

**>>> word.split(' ') *# Split on whitespace***

**['Hello', 'World']**

Startswith / Endswith

**word = "hello world"**

**>>> word.startswith("H")**

**True**

**>>> word.endswith("d")**

**True**

**>>> word.endswith("w")**

**False**

Repeat Strings

**print "."\* 10 *# prints ten dots***

**>>> print "." \* 10**

**..........**

Replacing

**word = "Hello World"**

**>>> word.replace("Hello", "Goodbye")**

**'Goodbye World'**

Changing Upper and Lower Case Strings

**string = "Hello World"**

**>>> print string.upper()**

**HELLO WORLD**

**>>> print string.lower()**

**hello world**

**>>> print string.title()**

**Hello World**

**>>> print string.capitalize()**

**Hello world**

**>>> print string.swapcase()**

**hELLO wORLD**

Reversing

**string = "Hello World"**

**>>> print ' '.join(reversed(string))**

**d l r o W o l l e H**

Strip

Python strings have the strip(), lstrip(), rstrip() methods for removing

any character from both ends of a string.

If the characters to be removed are not specified then white-space will be removed

**word = "Hello World"**

Strip off newline characters from end of the string

**>>> print word.strip('**

**')**

**Hello World**

**strip() *#removes from both ends***

**lstrip() *#removes leading characters (Left-strip)***

**rstrip() *#removes trailing characters (Right-strip)***

**>>> word = " xyz "**

**>>> print word**

**xyz**

**>>> print word.strip()**

**xyz**

**>>> print word.lstrip()**

**xyz**

**>>> print word.rstrip()**

**xyz**

Concatenation

To concatenate strings in Python use the "+" operator.

**"Hello " + "World" *# = "Hello World"***

**"Hello " + "World" + "!"*# = "Hello World!"***

Join

**>>> print ":".join(word) *# #add a : between every char***

**H:e:l:l:o: :W:o:r:l:d**

**>>> print " ".join(word) *# add a whitespace between every char***

**H e l l o W o r l d**

Testing

A string in Python can be tested for truth value.

The return type will be in Boolean value (True or False)

**word = "Hello World"**

**word.isalnum() *#check if all char are numbers***

**word.isalpha() *#check if all char in the string are alphabetic***

**word.isdigit() *#test if string contains digits***

**word.istitle() *#test if string contains title words***

**word.isupper() *#test if string contains upper case***

**word.islower() *#test if string contains lower case***

**word.isspace() *#test if string contains spaces***

**word.endswith('d') *#test if string endswith a d***

**word.startswith('H') *#test if string startswith H***

34. how to copy a dictonary in python?

*By "shallow copying" it means the*content*of the dictionary is not copied by value, but just creating a new reference*.

>>> a = {1: [1,2,3]}

>>> b = a.copy()

>>> a, b

({1: [1, 2, 3]}, {1: [1, 2, 3]})

>>> a[1].append(4)

>>> a, b

({1: [1, 2, 3, 4]}, {1: [1, 2, 3, 4]})

*In contrast, a deep copy will copy all contents by value.*

>>> c = copy.deepcopy(a)

>>> a, c

({1: [1, 2, 3, 4]}, {1: [1, 2, 3, 4]})

>>> a[1].append(5)

>>> a, c

({1: [1, 2, 3, 4, 5]}, {1: [1, 2, 3, 4]})

*So:*

1. a = b: *Reference assignment, Make a and b points to the same object.*
2. a ---,
3. v
4. {1: L}
5. ^ |

b ---' '----> [1,2,3]

1. a = b.copy(): *Shallow copying, a and b will become two isolated objects, but their contents still share the same reference*
2. a ---> {1: L}
3. |
4. >---> [1,2,3]
5. |

b ---> {1: M}

1. a = copy.deepcopy(b): *Deep copying, a and b's structure and content become completely isolated.*
2. a ---> {1: L}
3. ‘-----> [1,2,3]
4. b ---> {1: M}

‘-----> [1,2,3]

57.what Is pass stmt?

*It is used when a statement is required syntactically but you do not want any command or code to execute.*

### ***Example***

#!/usr/bin/python

for letter in 'Python':

if letter == 'h':

pass

print 'This is pass block'

print 'Current Letter :', letter

print "Good bye!"

*When the above code is executed, it produces following result −*

Current Letter : P

Current Letter : y

Current Letter : t

This is pass block

Current Letter : h

Current Letter : o

Current Letter : n

Good bye!

101.how to create a directory?

import os, sys

# Path to be created

path = "/tmp/home/monthly/daily/hourly"

if not os.path.exists(path):

os.mkdir( path, 0755 );

103.how to change directories?

import os

path = "/usr/tmp"

# Check current working directory.

retval = os.getcwd()

print "Current working directory %s" % retval

# Now change the directory

os.chdir( path )

# Check current working directory.

retval = os.getcwd()

104.how to display current working directories?

import os

# Check current working directory.

retval = os.getcwd()

105.how to delete the directory?

import shutil

shutil.rmtree('/folder\_name')

122.difference b/w pass and continue?

*The pass keyword is a no-op keyword. It does exactly nothing. It's often used as a placeholder for code which will be added later:*

if response == "yes":

pass # process "yes" case

*The continue keyword, on the other hand, is used to restart a loop at the control point, such as with:*

for i in range(10):

if i % 2 == 0:

continue

print i

*That loop will only output the odd numbers since continue returns to the loop control statement (for) for iterations where i is even.*

*In terms of an empty for loop, they're functionally identical. We can use either of:*

for i in range(10):

pass

for i in range(10):

continue

74.what is filter?

75.what is map?

76.what is reduce?

>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]  
>>>   
>>> print filter(lambda x: x % 3 == 0, foo)

|  |
| --- |
| [18, 9, 24, 12, 27] |

>>>   
>>> print map(lambda x: x \* 2 + 10, foo)

|  |
| --- |
| [14, 46, 28, 54, 44, 58, 26, 34, 64] |

>>>   
>>> print reduce(lambda x, y: x + y, foo)

|  |
| --- |
| 139 |

*In the first example, filter() calls our lambda function for each element of the list, and returns a new list that contains only those elements for which the function returned "True". In this case, we get a list of all elements that are multiples of 3. The expression x % 3 == 0 computes the remainder of x divided by 3 and compares the result with 0 (which is true if x is evenly divisible by 3).*

*In the second example, map() is used to convert our list. The given function is called for every element in the original list, and a new list is created which contains the return values from our lambda function. In this case, it computes 2 \* x + 10 for every element.*

*Finally, reduce() is somewhat special. The "worker function" for this one must accept two arguments (we've called them x and y here), not just one. The function is called with the first two elements from the list, then with the result of that call and the third element, and so on, until all of the list elements have been handled.*

81.what is difference b/w append and extend?

*If we append a list to another list, we add the new list as a single extra list to the original, thus making the original list just one longer with an item that is itself a list. But if we extend a list with another list, we add each element of the new list onto the original. Also we cannot extend a list with a single integer. The parameter of the extend method should be an iterable.*

>>> list1=[1,2,3,4,5]

>>> list2=[6,7,8,9,0]

>>> list1.append(6)

>>> print list1

[1, 2, 3, 4, 5, 6]

>>> list2.extend(2)

Traceback (most recent call last):

File "<pyshell#52>", line 1, in <module>

list2.extend(2)

TypeError: 'int' object is not iterable

>>> list2.extend([2])

>>> print list2

[6, 7, 8, 9, 0, 2]

28.what is eval and exec statement?

*Basically, [eval](https://docs.python.org/3/library/functions.html" \l "eval) is used to****eval****uate a single dynamically generated Python expression, and*[*exec*](https://docs.python.org/3/library/functions.html#exec)*is used to****exec****ute dynamically generated Python code only for its side effects.*

*eval and exec have these two differences:*

1. *eval accepts only a****single expression****, exec can take a code block that has Python statements: loops, try: except:, class and function/method definitions and so on.*

*An expression in Python is whatever you can have on the right-hand side of a variable assignment:*

1. *eval****returns the value****of the given expression, whereas exec ignores the return value from its code, and always returns None (in Python 2 it is a statement and cannot be used as an expression, so it really does not return anything).*

>>> a = 5

>>> eval('37 + a') # it is an expression

42

>>> exec('37 + a') # it is an expression statement

>>> exec('a = 47') # modify a global variable as a side effect

>>> a

47

>>> eval('a = 47') # you cannot evaluate a statement

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<string>", line 1

a = 47

^

SyntaxError: invalid syntax

21.how to convert upcase letter in lower case letter?

*Use* .lower() *method*

22.how to remove tailing white spaces?

>>> s = ' abdcde '

*Remove trailing whitespace*

>>> s.rstrip()

' abdcde'

*Remove leading whitespace*

>>> s.lstrip()

'abdcde '

*Remove all whitespace*

>>> s.strip()

'abdcde'

23.what is zfill()?

***zfill(width)*** *is zero fill in the left of the string until the string reaches to the length specified by width*

96.how to open a file?

*Before you can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object, which would be utilized to call other support methods associated with it.*

### *Syntax*

file object = open(file\_name [, access\_mode][, buffering])

*Here are parameter details:*

* ***file\_name****: The file\_name argument is a string value that contains the name of the file that you want to access.*
* ***access\_mode****: The access\_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).*
* ***buffering****: If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).*

|  |  |
| --- | --- |
| ***Modes*** | ***Description*** |
| *r* | *Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.* |
| *rb* | *Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.* |
| *r+* | *Opens a file for both reading and writing. The file pointer placed at the beginning of the file.* |
| *rb+* | *Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.* |
| *w* | *Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.* |
| *wb* | *Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.* |
| *w+* | *Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.* |
| *wb+* | *Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.* |
| *a* | *Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.* |
| *ab* | *Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.* |
| *a+* | *Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.* |
| *ab+* | *Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.* |

121.what is pprint?

*The [pprint](https://docs.python.org/2/library/pprint.html" \l "module-pprint" \o "pprint: Data pretty printer.) module provides a capability to “pretty-print” arbitrary Python data structures in a form which can be used as input to the interpreter. If the formatted structures include objects which are not fundamental Python types, the representation may not be loadable. This may be the case if objects such as files, sockets, classes, or instances are included, as well as many other built-in objects which are not representable as Python constants.*

*The formatted representation keeps objects on a single line if it can, and breaks them onto multiple lines if they don’t fit within the allowed width.*