Programming Course

Day - Two

Data Types, Expressions

Day - 2 Agenda

- Error Handling
- Strings
- Lists
- Dictionaries
- Sets
- Tuples
- Variables Expressions
- Mathematical Expressions
- Recording and processing Input from user (input)
- Iterators
- Collections
- Inbuilt functions
  - In, with, range

Handling Errors

Two Types of Errors - [1] Syntax Error [2] Exceptions

Syntax Errors - Correct the Syntax!

Exceptions - Handle Exceptions, depends on the Logic of Application. Depends on Data of Application

IOError : If the file cannot be opened.

ImportError : If python cannot find the module

ValueError Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value

KeyboardInterrupt Raised when the user hits the interrupt key (normally Control-C or Delete)

EOFError Raised when one of the built-in functions (input() or raw_input()) hits an end-of-file condition (EOF) without reading any data

Handling Errors(Cont..)

```
>>> try:
...     d = 1/0
... except:
...     print ("I could not divide 1 by 0")
...
I could not divide 1 by 0
>>>
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> try:
...     d = 1/0
... except ZeroDivisionError:
...    print("Don't try to divide by 0")
...
Don't try to divide by 0
>>> try:
...     d = 10/2
... except ZeroDivisionError:
...    print("Don't try to divide by 0")
... else:
...    print("I am in else")
...
I am in else
>>> print(d)
5.0
>>>
```

Task - 1

Develop a script to take two numbers as input. Divide first number by second number. It should handle division by zero. The interaction of the script should look something similar to following

$divide.py

Enter First Number:10

Enter Second Number:0

You tried to divide 10 by zero

$*divide.py*

Enter First Number:10

Enter Second Number:2

The division is: 5.0

Collections

- A collection is similar to a basket that you can
  - add items
  - remove items
- The items can be same types or different types - it depends on collecton

In some sense - Collection is storage construct that allows you to collect things

Python offers several built-in types that fall under a vague category called collections. We will talk about following:

Strings	**Lists**
Dictionaries	**Sets**

Strings

- Strings are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:])
    - with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

OPERATIONS

- Concatenation +
- Repetition with the help of asterisk *

Strings are immutable

Strings

A string is a sequence of characters. A character is simply a symbol. For example, the English language has 26 characters. The same symbols - computer stores in numbers internally (0's and 1's) (encoding/decoding process - ASCII, Unicode)

```
my_string = 'Hello'
print(my_string)

my_string = "Hello"
print(my_string)

my_string = '''Hello'''
print(my_string)

# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
           the world of Python"""
print(my_string)
```

strings_one.py

```
Hello
Hello
Hello
Hello, welcome to
           the world of Python
```

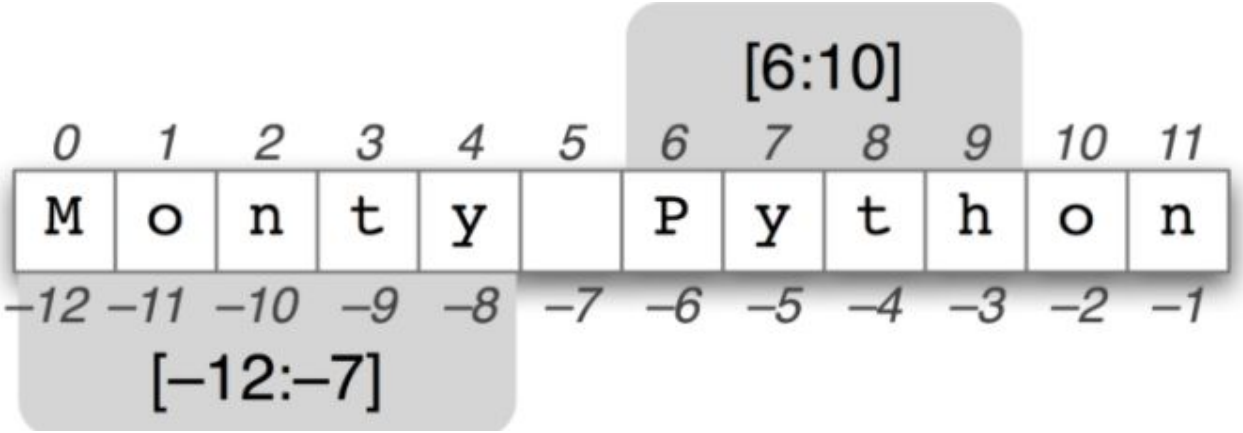
Strings (Cont ...)

```
#!/usr/bin/python3

str = 'Hello World!'

print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])      # Prints string starting from 3rd character
print (str * 2)      # Prints string two times
print (str + "TEST") # Prints concatenated string
```

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

strings_demo.py

Strings (Cont...)




```
#!/usr/bin/python3

s = 'Hello World'
print (len(s))
print (type(s))

s = 'Monty Python'

print(s[6:10])
print(s[-12:-7])
```

```
11
<class 'str'>
Pyth
Monty
```

strings_len.py

Strings - Important Methods

- s.lower(), s.upper() -- returns the lowercase or uppercase version of the string
- s.strip() -- returns a string with whitespace removed from the start and end
- s.isalpha()/s.isdigit()/s.isspace()... -- tests if all the string chars are in the various character classes
- s.startswith('other'), s.endswith('other') -- tests if the string starts or ends with the given other string
- s.find('other') -- searches for the given other string (not a regular expression) within s, and returns the first index where it begins or -1 if not found
- s.replace('old', 'new') -- returns a string where all occurrences of 'old' have been replaced by 'new'
- s.split('delim') -- returns a list of substrings separated by the given delimiter. T
- s.join(list) -- opposite of split()

Quiz

What is the output of following code?

1
```
>>> "a" + "bc"
```

2
```
>>> "abcd"[2]
```

3
```
>>> print(r"\nhello")
```

4
```
>>> str1 = 'helloworld'
>>> str1[::-1]
```

5
```
>>> jio = '4G Liyo To Jio!'
>>> jio[3:7]
```

6
```
>>> jio = '4G Liyo To Jio!'
>>> jio[7:]
```

String Formatting - Text Displaying

```
# implicit order (default one)
default_order = "{}, {} and {}".format('Hari','Sadu','Naukari')
print('\n--- Default Order ---')
print(default_order)

# order using positional argument
positional_order = "{1}, {0} and {2}".format('Hari','Sadu','Naukari')
print('\n--- Positional Order ---')
print(positional_order)

# order using keyword argument
keyword_order = "{s}, {n} and {h}".format(h='Hari',s='Sadu',n='Naukari')
print('\n--- Keyword Order ---')
print(keyword_order)
```

```
--- Default Order ---
Hari, Sadu and Naukari

--- Positional Order ---
Sadu, Hari and Naukari

--- Keyword Order ---
Sadu, Naukari and Hari
```

String Format (Cont ...)

```
# formatting integers
#'Binary representation of 12 is 1100'
print("Binary representation of {0} is {0:b}".format(12))

 # formatting floats
#'Exponent representation: 1.566345e+03'
print("Exponent representation: {0:e}".format(1566.345))

# round off
#'One third is: 0.333'
print( "One third is: {0:.3f}".format(1/3))

# string alignment
#'|butter    |  bread   |       ham|'
print( "|{:<10}|{:^10}|{:>10}|".format('butter','bread','ham'))
```

```
Binary representation of 12 is 1100
Exponent representation: 1.566345e+03
One third is: 0.000
|butter    |  bread   |       ham|
```

Python Lists

A list contains items separated by commas and enclosed within square brackets ([])

```
#!/usr/bin/python3

list      = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']


print (list )              # Prints complete list
print (list[0]   )         # Prints first element of the list
print (list[1:3] )         # Prints elements starting from 2nd till 3rd
print (list[2:] )          # Prints elements starting from 3rd element
print (tinylist * 2 )      # Prints list two times
print (list + tinylist )   # Prints concatenated lists

print (type(list))         # type
print(len(list))           # length
```

List is sequence

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
<class 'list'>
5
```

lists_demo.py

for - Loop

```
#!/usr/bin/python3

languages = ["C", "Java", "Perl", "Python", "Node.js", "JavaScript"]

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

```
C
Java
Perl
Python
Node.js
JavaScript
```

A few important key words:

- break
- continue

for_demo.py

for - loop (Cont...)

```
#!/usr/bin/python3

edibles = ["ham", "spam","eggs","nuts"]
for food in edibles:
    if food == "spam":
        print("No more spam please!")
        break
    print("Great, delicious " + food)
else:
    print("I am so glad: No spam!")
print("Finally, I finished stuffing myself")
```

```
Great, delicious ham
No more spam please!
Finally, I finished stuffing myself
```

for_demo_adv.py

for_demo_adv_nospam.py

while - loop

```
#!/usr/bin/python3

count = 0
while (count < 9):
   print ('The count is:', count)
   count = count + 1

print ("Good bye!")
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

while_demo.py

Quiz

What is the output of the following code?

```
#!/usr/bin/python3
presidents = ['Clinton', 'Barack', 'Trump']
for president in presidents:
   if president == 'Trump':
        print ("No more Trump - please")
        break
   print ("Great - President: ", president)
else:
   print ("I am so glad - even Trump was covered")

print ("I am done")
```

Option - A

```
Great - President:  Clinton
Great - President:  Barack
No more Trump - please
I am done
```

Option - B

```
Great - President:  Clinton
Great - President:  Barack
Great - President:  Trump
I am so glad - even Trump was covered
I am done
```

Quiz

Study the following Script. The script stops, makes exit as soon as User enters Char "Q". Is that True?

```
#!/usr/bin/python3

ch = True
while (ch):
   key = input("Enter Key: ")
   print ("You entered: ", key)
   if key == 'Q':
       ch = False
       print ("Oh - you entered Q, you are making exit")
```

Built-in Method - “range”

```
>>> for i in range(5):
...     print (i)
...
0
1
2
3
4
```

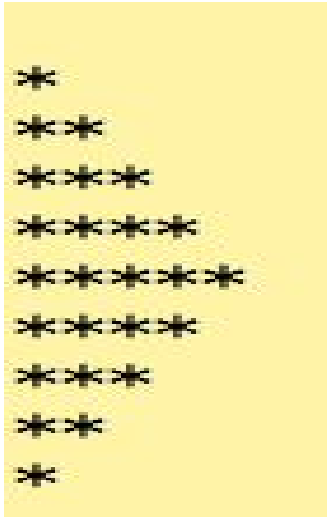
```
>>> for i in range(2, 5):
...     print (i)
...
2
3
4
```

```
>>> for i  in range(3, 10, 2):
...     print (i)
...
3
5
7
9
>>>
```

```
>>> L  = list(range(10))
>>> print (L)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(10)
range(0, 10)
>>> type(range(10))
<class 'range'>
```

Task - 2

Write a script to print pattern like below.



Task - 3

Write a program - to count “even” and “odd” whole numbers (0, 1, 2... i.e. zero and all positive integers) which are less than 50.

Revisit Lists

Accessing Lists

- n = len(L)
- item = L[index]
- seq = L[start:stop]
- seq = L[start:stop:step]
- seq = L[::2] # get every other item, starting with the first
- seq = L[1::2] # get every other item, starting with the second

Revisit Lists (Cont ...)

```
#!/usr/bin/python3
L = ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
print (len(L))

item = L[5]

print (item)

seq = L[2:7]

print (seq)

seq_step = L[1:8:2]
print (seq_step)

seq_trick = L[::2]
print (seq_trick)
```

lists_one.py

Lists (Cont...)

```
for item in L:
    print (item)
```

```
for index, item in enumerate(L):
        print index, item
```

Lists (Cont...)

Other important Operations

- L.append(item)
- L.extend(sequence)
- L.insert(index, item)
- del L[i]
- del L[i:j]
- item = L.pop() # last item
- item = L.pop(0) # first item
- item = L.pop(index)
- L.remove(item)
- L.reverse()
- L.sort()
- out = sorted(L)

Lists (Cont ...)

```
#!/usr/bin/python3

lst = ["easy", "simple", "cheap", "free"]
print (lst[-1])


lst = [3, 5, 7]
lst.append(42)

print (lst)

lst = lst.append(42)
print (lst)

cities = ["Pune", "Mumbai", "Nagpur", "Nashik"]
c1 = cities.pop(0)
print (c1)
print (cities)

c2 = cities.pop(1)
print (c2)
print (cities)
```

```
free
[3, 5, 7, 42]
None
Pune
['Mumbai', 'Nagpur', 'Nashik']
Nagpur
['Mumbai', 'Nashik']
```

lists_two.py

Lists (Cont ...)

```
#!/usr/bin/python3
L = [1, 10, 20, 5, 15, 9, 19]
print (L)

L.reverse()
print (L)

L.sort()
print (L)

L = [1, 10, 20, 5, 15, 9, 19]
print (L)

NEW_L = sorted(L)
print (NEW_L)
print (L)
```

```
[1, 10, 20, 5, 15, 9, 19]
[19, 9, 15, 5, 20, 10, 1]
[1, 5, 9, 10, 15, 19, 20]
[1, 10, 20, 5, 15, 9, 19]
[1, 5, 9, 10, 15, 19, 20]
[1, 10, 20, 5, 15, 9, 19]
```

lists_three.py

Task - 4

Develop a script “factors_list.py” - take the input number from user. Say this number is “N”.

Print the list with “N+1” elements where each element is list of factors for the the index.

```
$ ./factors_list.py
Enter the Number: 5
[[], [1], [1, 2], [1, 3], [1, 2, 4], [1, 5]]
$ ./factors_list.py
Enter the Number: 10
[[], [1], [1, 2], [1, 3], [1, 2, 4], [1, 5], [1, 2, 3, 6], [1, 7], [1, 2, 4, 8], [1, 3, 9], [1, 2, 5, 10]]
$
```

Quiz

What is the output of following code?

```
#!/usr/bin/python3

names1 = ['Amir', 'Sharukh', 'Chales', 'Dao']
names2 = names1
names3 = names1[:]

names2[0] = 'Alice'
names3[1] = 'Bob'

sum = 0
for ls in (names1, names2, names3):
     if ls[0] == 'Alice':
          sum += 1
     if ls[1] == 'Bob':
          sum += 10

print (sum)
```

Quiz

1. What is the output of following code?

```
>>> names = ['Amir', 'Sahrukh', 'Chales', 'Dao']
>>> print (names[-1][-1])
```

2. Can this code run properly? Any error you expect from code?

```
>>> names = ['Amir', 'Sahrukh', 'Chales', 'Dao']
>>> print (names[-1][-1000])
```

Quiz

What gets printed after execution of following code?

1

```
>>> names = ['Amir', 'Sahrukh', 'Chales', 'Dao']
>>> loc = names.index('Edward')
```

2

```
>>> names = ['Amir', 'Sahrukh', 'Chales', 'Dao']
>>> if 'Amir' in names:
...     print (1)
... else:
...     print (2)
...
```

3

```
>>> numbers = [1, 2, 3, 4]
>>> numbers.append([5, 6, 7, 8])
>>> print (len(numbers))
```

Revisit Strings

```
>>> a = "this is a string"
>>> print (a)
this is a string
>>> a.split()
['this', 'is', 'a', 'string']
>>> b = a.split()
>>> type(b)
<class 'list'>
>>> b
['this', 'is', 'a', 'string']
>>>
```

```
>>> user = 'nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false'
>>> fields = user.split(':')
>>> print (fields)
['nobody', '*', '-2', '-2', 'Unprivileged User', '/var/empty', '/usr/bin/false']
>>>
```

Revisit Strings (Cont …)

```
>>> u = ":".join(fields)
>>> print (u)
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
>>> type(u)
<class 'str'>
```

```
>>> b
['this', 'is', 'a', 'string']
>>> a = " ".join(b)
>>> print (a)
this is a string
```

Task - 5

You are given a string. Split the string on a `" "` (space) delimiter and join using a `-` hyphen. The script name should be: “split_join.py”

Sample Input

```
Hello to Python Programming Course
```

Sample Output

```
Hello-to-Python-Programming-Course
```

Python Dictionaries

They are build-in in Python - Also known as:

- Associate Array
- Map
- Hash Map
- Un-ordered Map

Contains a series of key, value mappings where the "key" is of any type that is *hashable.*

The "value" may be of any type and value types need not be homogeneous

Dictionaries (Cont …)

```
#!/usr/bin/python3

d1 = {}
print ("----------I")
print (type(d1))

d2 = {'one': 1, 'two':2}
print ("----------II")
print (d2)
print ("----------III")
print (type(d2))

d3 = dict(one=2, three=4)
print ("----------IV")
print (d3)
print (type(d3))

d4 = dict([(1, 2), (3, 4)])
print ("----------V")
print (d4)

d5 = dict({1:2, 3:4})
print ("----------V")
print (d5)
```

```
----------I
<class 'dict'>
----------II
{'one': 1, 'two': 2}
----------III
<class 'dict'>
----------IV
{'one': 2, 'three': 4}
<class 'dict'>
----------V
{1: 2, 3: 4}
----------V
{1: 2, 3: 4}
```

dictionaries.py

Dictionaries (Cont...)

Remember:

If you're searching for a value in a dictionary and you use a `for` loop, you're doing it wrong.

In dictionaries we can find a value instantly, without needing to search through the whole dictionary manually, using the form

`value = my_dict['key']`

or

`value = my_dict.get('key', None)`.

Dictionaries (Cont …)

- d.clear()
- d.copy()
- del k[d]
- dict.fromkeys(seq[, value])
- iteration/accessing elements of dictionaries
- for key in my_dictionary:
- for key, value in my_dictionary.items():
- for value in my_dictionary.values():
- iter(d)
- len(d)
- d.keys()
- d.values()
- d.items()

Dictionaries (Cont …)

```
>>> my_dict = {'name':'Jack', 'age': 26}
>>> print(my_dict['name'])
Jack
>>> print(my_dict.get('age'))
26
>>> my_dict.get('address')
>>> my_dict['address']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'address'
>>>
```

```
>>> my_dict = {'name':'Jack', 'age': 26}
>>>
>>> my_dict['age'] = 27
>>>
>>> print(my_dict)
{'name': 'Jack', 'age': 27}
>>> my_dict['address'] = 'Downtown'
>>> print(my_dict)
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
>>>
```

Dictionaries (Cont …)

```
>>> squares = {1:1, 2:4, 3:9, 4:16, 5:25}
>>> print(squares.pop(4))
16
>>> squares
{1: 1, 2: 4, 3: 9, 5: 25}
>>>
>>>
>>> print(squares.popitem())
(1, 1)
>>> print(squares)
{2: 4, 3: 9, 5: 25}
>>> del squares[5]
>>> print(squares)
{2: 4, 3: 9}
>>> squares.clear()
>>> print(squares)
{}
>>> del squares
>>> print(squares)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'squares' is not defined
>>>
```

Dictionaries (Cont ...)

```
>>> capitals = {'India':'Delhi', 'USA':'Washington',  'Japan':'Tokeyo',  'England':'London'}
>>> for k, v in capitals.items():
...     print (k, "-->", v)
...
India --> Delhi
England --> London
Japan --> Tokeyo
USA --> Washington
>>>
```

```
>>> for k in capitals.keys():
...     print (k, "-->", capitals[k])
...
India --> Delhi
England --> London
Japan --> Tokeyo
USA --> Washington
```

Dictionaries (Cont ...)

```
>>> capitals.keys()
dict_keys(['India', 'England', 'Japan', 'USA'])
>>> list(capitals.keys())
['India', 'England', 'Japan', 'USA']
>>>
```

```
>>> for k in capitals:
...     print (k)
...
India
England
Japan
USA
```

Quiz

What is the output of following code?

```
#!/usr/bin/python3

confusion = {}
confusion[1] = 1
confusion['1'] = 2
confusion[1] += 1

sum = 0
for k in confusion:
    sum += confusion[k]

print(sum)
```

Quiz

What is the output of following code?

```
#!/usr/bin/python3
confusion = {}
confusion[1] = 1
confusion['1'] = 2
confusion[1.0] = 4

sum = 0
for k in confusion:
    sum += confusion[k]

print(sum)
```

Quiz

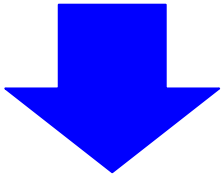
```
#!/usr/bin/python3

boxes = {}
jars = {}
crates = {}

boxes['cereal'] = 1
boxes['candy'] = 2
jars['honey'] = 4
crates['boxes'] = boxes
crates['jars'] = jars

print(len(crates[boxes]))
```

Task - 6

- Create a List - "A", elements are integers. List may have repeated elements
- Create new "B", which has same element from "A" but all elements of "B" are Unique Elements
- Develop a function - to take generic list and return sorted, unique element list (we are still talking about integers)

Caching or Memoization

```
def some_function (arg1, arg2, arg3,..., argN):
        result = some computation involving arg1, arg2, arg3, ... argN
        return result
```



```
catche = {}
def some_function_with_catching(arg1, arg2, arg3, ..., argN):
        key = str(arg1) + str(arg2) + str(arg3) + .... + str(argN)
        if key in cache:
              return catche[key]
        else:
             result = same computation involving arg1, arg2, arg3, ..., argN
             catche[key]=result
             return result
```

Fibonacci Numbers

The Fibonacci Sequence is the series of numbers:

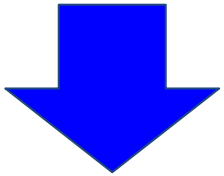
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

The next number is found by adding up the two numbers before it.

- The 2 is found by adding the two numbers before it (1+1)
- The 3 is found by adding the two numbers before it (1+2),
- And the 5 is (2+3),
- and so on!

Caching or Memoization - Fibonacci Numbers

```
def fib(n):
    return n if n < 2 else fib(n-2) + fib(n-1)
```



```
__fib_cache = {}
def fib(n):
    if n in __fib_cache:
        return __fib_cache[n]
    else:
        __fib_cache[n] = n if n < 2 else fib(n-2) + fib(n-1)
        return __fib_cache[n]
```

Sets

- A set is an unordered collection of items.
- Every element is unique (no duplicates)
  - must be immutable (which cannot be changed).
- However, the set itself is mutable. We can add or remove items from it.

Sets (Cont …)

```
#!/usr/bin/python3

# Create a set.
items = {"arrow", "spear", "arrow", "arrow", "rock"}

print (type(items))
print(items)
print(len(items))

if "rock" in items:
    print("Rock exists")

if "clock" not in items:
    print("Cloak not found")
```

```
<class 'set'>
{'arrow', 'rock', 'spear'}
3
Rock exists
Cloak not found
```

Set_one.py

Set (Cont ...)

Built-in "set" Methods help convert other data-types into set

```
>>> s = set(["Perl", "Python", "PHP"])
>>> type(s)
<class 'set'>
>>> s
{'Perl', 'Python', 'PHP'}
>>>
```

Sets doesn't allow mutable objects

```
>>> s = set(["Perl", "Python", "PHP", ["node", "javascript"]])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>>
```

Sets (Cont ...)

```
>>> colors = {"red","green"}
>>> colors.add("yellow")
>>> colors
{'yellow', 'green', 'red'}
```

```
>>> cities = {"Mumbai", "Pune"}
>>> cities
{'Pune', 'Mumbai'}
>>> cities.clear()
>>> cities
set()
```

```
>>> maha_cities = {"Mumbai", "Pune"}
>>> big_cities = maha_cities.copy()
>>> maha_cities
{'Pune', 'Mumbai'}
>>> big_cities
{'Pune', 'Mumbai'}
>>> maha_cities.clear()
>>> maha_cities
set()
>>> big_cities
{'Pune', 'Mumbai'}
```

Sets (Cont ...)

Assignment is bad idea for copy :)

```
>>> maha_cities = {"Mumbai", "Pune"}
>>> new_cities = maha_cities
>>> new_cities
{'Pune', 'Mumbai'}
>>> maha_cities.clear()
>>> new_cities
set()
>>> maha_cities
set()
```

```
>>> x = {"a","b","c","d","e"}
>>> y = {"b","c"}
>>> z = {"c","d"}
>>> x.difference(y)
{'e', 'a', 'd'}
```

```
>>> x = {"a","b","c","d","e"}
>>> y = {"b","c"}
>>> x - y
{'e', 'a', 'd'}
```

```
>>> x = {"a","b","c","d","e"}
>>> y = {"b","c"}
>>> x.difference_update(y)
>>> x
{'a', 'd', 'e'}
```

Set (Cont ...)

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d","e","f","g"}
>>> x.union(y)
{'f', 'g', 'e', 'c', 'a', 'd', 'b'}
>>>
```

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d","e","f","g"}
>>> x.intersection(y)
{'e', 'c', 'd'}
>>>
```

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d","e","f","g"}
>>> x|y
{'f', 'g', 'e', 'c', 'a', 'd', 'b'}
```

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d","e","f","g"}
>>> x & y
{'e', 'c', 'd'}
```

Set (Cont …)

```
>>> x = {"a","b","c"}
>>> y = {"c","d","e"}
>>> x.isdisjoint(y)
False
>>>
>>>
>>> x = {"a","b","c"}
>>> y = {"d","e","f"}
>>> x.isdisjoint(y)
True
```

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d"}
>>> x.issubset(y)
False
>>> y.issubset(x)
True
```

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d"}
>>> x > y
True
>>> x <y
False
>>> x <= y
False
>>> x < x
False
>>> x <= x
True
```

Set (Cont …)

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d"}
>>> x.issuperset(y)
True
>>> x >y
True
>>> x >= y
True
>>> x >= x
True
```

```
>>> x = {"a", "b"}
>>> x.pop()
'a'
>>> x.pop()
'b'
>>> x.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'pop from an empty set'
>>>
```

Quiz

What is the output of following code?

1.
```
>>> len({'a', 'b', 'c', 'a', 'b', 'c'})
```

2.
```
>>> sum({1, 2, 3, 4, 5, 1, 2, 4, 5})
```

3.
```
>>> run_times = {10.03, 9.69, 9.58, 9.82, 9.76, 9.63}
>>> min(run_times)
```

Tupples

A tuple is a sequence of immutable Python objects.

Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

```
>>> tup1 = (12, 34.56);
>>> tup1
(12, 34.56)
>>> type(tup1)
<class 'tuple'>
>>> tup1[0] = 100;
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
>>> tup2 = ('abc', 'xyz');
>>> tup1 + tup2
(12, 34.56, 'abc', 'xyz')
```

Tuple (Cont …)

```
>>> tup = ('physics', 'chemistry', 1997, 2000);
>>>
>>> del tup
>>> print (tup)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'tup' is not defined
>>>
```

Operations below work on Tuples:

- Concatenation
- Repetition
- Members (3 in (1, 2, 3) evaluates to True)
- len
- Iteration (for e in (1,2, 3):) works best

Indexing, Slicing works similar to other sequences!

Iterators

In computer programming, an **iterator** is an object
that enables a programmer to traverse a container (examples of containers are: list, dictionaries)

We already saw some - sequences like: Lists, Strings, Tuples. They have indexes
So iterating through them was easy (0 to len() - use for loop)
We saw some non-sequences like Set, Dictionaries. They do not have indexes.
So iteration is bit tedious! Off-course they do have built-in “iter” method.

Iterator (Cont …)

An **iterable** is any object that **you can loop over with a for loop**. They need not have finite elements, need not have indexes (e.g dictionaries)

```
>>> from itertools import count
>>> multiple_of_five = count(step=5)
>>>
>>> for n in multiple_of_five:
...     if n> 50:
...          break
...     print (n)
...
0
5
10
15
20
25
30
35
40
45
50
>>>
```

The script will run forever if you remove “break” statement from this code

Iterator (Cont ...)

All iterables can be passed to the built-in iter function to get an **iterator** from them.

```
>>> iter(['hello', 'bye'])
<list_iterator object at 0x7fa0f5b90470>
>>> iter({'India':'Delhi', 'USA':'Washington',  'Japan':'Tokeyo',  'England':'London'} )
<dict_keyiterator object at 0x7fa0f5b92408>
>>> iter({"Usain Bolt", "Sara Moreira", "Lalita Babar"})
<set_iterator object at 0x7fa0f5b99dc8>
>>> iter("Welcome to Python Class")
<str_iterator object at 0x7fa0f5c10400>
>>>
```

Iterator (Cont …)

```
>>> atk = iter(['hello', 'bye'])
>>> next(atk)
'hello'
>>> next(atk)
'bye'
>>> next(atk)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

```
>>> atk = iter(['hello', 'bye'])
>>> atk.__next__()
'hello'
>>> atk.__next__()
'bye'
>>> atk.__next__()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

Iterator (Cont …)

```
def print_each(iterable):
    iterator = iter(iterable)
    while True:
        try:
            item = next(iterator)
        except StopIteration:
            break  # Iterator exhausted: stop the loop
        else:
            print(item)
```

Collections - Counter

A Counter is a container that keeps track of how many times equivalent values are added.

Let us look at initialization

```
>>> import collections
>>> print(collections.Counter(['a', 'b', 'c', 'a', 'b', 'b']))
Counter({'b': 3, 'a': 2, 'c': 1})
>>> print(collections.Counter({'a': 2, 'b': 3, 'c': 1}))
Counter({'b': 3, 'a': 2, 'c': 1})
>>> print(collections.Counter(a=2, b=3, c=1))
Counter({'b': 3, 'a': 2, 'c': 1})
>>>
```

Collections - Counter (Cont ...)

```
>>> import collections
>>>
>>> c = collections.Counter()
>>> print('Initial :', c)
Initial : Counter()
>>>
>>> c.update('abcdaab')
>>> print('Sequence:', c)
Sequence: Counter({'a': 3, 'b': 2, 'c': 1, 'd': 1})
>>>
>>> c.update({'a': 1, 'd': 5})
>>> print('Dict    :', c)
Dict    : Counter({'d': 6, 'a': 4, 'b': 2, 'c': 1})
```

Collections - Counter (Cont ...)

```
>>> import collections
>>>
>>> c = collections.Counter('abcdaab')
>>>
>>> for letter in 'abcde':
...     print('{} : {}'.format(letter, c[letter]))
...
a : 3
b : 2
c : 1
d : 1
e : 0
```

Collections - Counter (Cont …)

- Order of elements - not guaranteed
- Elements with count <= 0 are ignored

```
>>> import collections
>>>
>>> c = collections.Counter('extremely')
>>> c['z'] = 0
>>> print(c)
Counter({'e': 3, 'm': 1, 'x': 1, 't': 1, 'l': 1, 'r': 1, 'y': 1, 'z': 0})
>>> print(list(c.elements()))
['e', 'e', 'e', 'm', 'x', 't', 'l', 'r', 'y']
>>> _
```

Collections - Counter (Cont …)

most_common() function

```
>>> import collections
>>> c = collections.Counter('extremelyextremelyextremelyextremelyextremelyextremelyextremely')
>>> print('Most common:')
Most common:
>>> for letter, count in c.most_common(2):
...     print('{}: {:>7}'.format(letter, count))
...
e:      21
m:       7
>>> for letter, count in c.most_common(3):
...     print('{}: {:>7}'.format(letter, count))
...
e:      21
m:       7
x:       7
```

Collections - Counter (Cont ...)

Arithmetic Operation on Counter

collections_counter_arithmetic.py

```
#!/usr/bin/python3

import collections

c1 = collections.Counter(['a', 'b', 'c', 'a', 'b', 'b', 'b', 't', 'a', 'a'])
c2 = collections.Counter('alphabet')

print('C1:', c1)
print('C2:', c2)

print('\nCombined counts:')
print(c1 + c2)

print('\nSubtraction:')
print(c1 - c2)

print('\nIntersection (taking positive minimums):')
print(c1 & c2)

print('\nUnion (taking maximums):')
print(c1 | c2)
```

Collections - Counter (Cont ...)

Arithmetic Operation on Counter

collections_counter_arithmetic.py

```
$ ./collections_counter_arithmetic.py
C1: Counter({'b': 4, 'a': 4, 'c': 1, 't': 1})
C2: Counter({'a': 2, 'b': 1, 'l': 1, 'h': 1, 't': 1, 'e': 1, 'p': 1})

Combined counts:
Counter({'a': 6, 'b': 5, 't': 2, 'l': 1, 'h': 1, 'e': 1, 'c': 1, 'p': 1})

Subtraction:
Counter({'b': 3, 'a': 2, 'c': 1})

Intersection (taking positive minimums):
Counter({'a': 2, 'b': 1, 't': 1})

Union (taking maximums):
Counter({'b': 4, 'a': 4, 'l': 1, 'h': 1, 't': 1, 'e': 1, 'c': 1, 'p': 1})
$
```

Zip - Iterate over multiple lists

```
>>> alist = ['a1', 'a2', 'a3']
>>> for i, a in enumerate(alist):
...         print (i, a)
...
0 a1
1 a2
2 a3
```

```
>>> for a, b in zip(alist, blist):
...       print (a, b)
...
a1 b1
a2 b2
a3 b3
>>>
```

Task-7

Q. Created List with even numbers and odd number - both less than 500. Demonstrate - iteration over both of them together. You need to use “zip” function from Python.

LAB Assignment

Problem 1. : Write a script to find the factors of any given number

Task - Name the script as - *number_factors.py* . The script takes number using **input().** The script creates the list of factors using remainder function and *for* loop.

$ number_factors.py

Enter Integer: 100

The factors of 100 are: [1, 2, 5, 10, 20, 25, 50, 100]

LAB Assignment (Cont ...)

Problem 2. : Write a script to find the Highest Common Factor (HCF) of any number of integers

Task - Name the script as - *hcf_numbers.py* . The script takes number using **input().** The script creates the list of factors using remainder function and *for* loop.

$ How many numbers: 3
Enter one number on each line below
5
10
15
The HCF of 5, 10, 15 are: 5

LAB Assignment (Cont ...)

Problem 3. Develop a Phonebook using Python Dictionaries

Name the script - *phone_book.py*.

```
$ phone_book.py
How many records you would like to enter in PhoneBook? : 2
Please enter those required (One record on each line, they should be space seperated).
Hari 984445454545
Sadu 23454544

Your records are entered in phonebook successfully. Now, please enter names which you would like to query?
(Once done - feel free to enter ctrl+D)

John
Hari
Sadu

Printing Phone Numbers for queries names:
For John - Number not found
Hari = 984445454545
Sadu - 23454544
```

LAB Assignment (Cont …)

Problem 4. Given number of integers, calculate and print the respective *mean*, *median*, and *mode* on separate lines. If your array contains more than One *modal value*, choose the numerically smallest one.

Sample Execution:

```
$ python3 mean_median_mode.py
10
10 23 45 23 45 67 89 23 45 45
Mean: 41.5
Median: 45.0
Mode: 45
```

Lab Assignment (Cont …)

Problem 5. Use **lab_two.py** assignment area to find:

-most active users (2 users)

-most active IP address (2 IP address)

-most viewed topics (2 topics)