Data Types and Operations

The data stored in memory can be of many types. For example, a person's name is stored as alphabets, age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has the following standard data types.

- Numbers
- String
- List
- Tuple
- Set
- Dictionary

1. Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example a = 1, b = 20. You can also delete the reference to a number object by using the **del** statement. The syntax of the **del** statement is as follows.

del variablel, variable2 , variable3,.....variableN

You can delete a single object or multiple objects by using the **del** statement. For example

del a

del a,b

Python supports four different numerical types.

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values) complex (complex numbers)
- complex (complex numbers)

Integers can of any length, it is only limited by the memory available. A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number. Complex numbers are written in form, **x+yj**, where x is the real part and y is the imaginary part.

Example Program

```
a,b,c,d=1,1.5,231456987,2+9j
print ("a=", a)
print("b=",b)
```

```
print("c=",c)
print("d=", d)
```

Output

```
a=1
b= 1.5
c=2314569870
d= (2+9j)
```

Mathematical Functions

Python provides various built-in mathematical functions to perform mathematic calculations. The following Table provides various mathematical functions and its purpose. For using the below functions, **all functions except abs(x), max(x1,x2,…, xn), min(x1,x2,... xn), round(x.n) and pow(x,y) need to import the math module** because these functions reside in the math module. The math module also defines two mathematical constants pi and e.

Function	Description
abs(x)	Returns the absolute value of x.
sqrt(x)	Finds the square root of x
ceil(x)	Finds the smallest integer not less than x.
floor(x)	Finds the largest integer not greater than x
pow(x,y)	Finds x raised to y.
exp(x)	Returns e^x, ie exponential of x.
fabs(x)	Returns the absolute value of x
log(x)	Finds the natural logarithm of x for >>0.
log10(x)	Finds the logarithm to the base 10 for x>0.
max(x1,x2,...xn)	Returns the largest of its arguments.
min(x1,x2,....xn)	Returns the smallest of its arguments.
round(x,n)	In case of decimal numbers, x will be rounded to n digits.
modf(x)	Returns the integer and decimal part as a tuple.

Example Program

```
import math
```

```
print("Absolute value of 120, abs(-120)))
print ("Square root of 25:"math.sqrt(25))
print("Ceiling of 12.2", math ceil (12.2))
print of 12.2.", math.floor(12.2))
print ("2 raised to 3:", pow(2,3))
print ("Exponential of 3,math.exp(3))
print("Absolute value of 123:", math.fabs(-123))
print("Natural Logarithm of 2:", math.log(2))
print("Logarithm to the Base 10 of 2: math. log10(2))
print ("Largest among 10, 4, 2:",max(10,4,2))
print("Smallest among 10.4, 2:",min(10,4,2))
print("12.6789 rounded to 2 decimal places:", round (12.6789,2))
print ("Decimal part and integer part of 12.090876:", math.modf(12.090876))
```

Output

```
Absolute value of -120:         120
Square root of 25:      5.0
Ceiling of 12.2:        13
Floor of 12.2:          12
2 raised to 3:          8
Exponential of 3:       20.085536923187668
Absolute value of -123:         123.0
Natural Logarithm of 2:          0.6931471805599453
Logarithm to the Base 10 of 2: 0.3010299956639812
Largest among 10, 4, 2:          10
Smallest among 10,4, 2:         2
12.6789 rounded to 2 decimal places: 12.68
Decimal part and integer part of 12.090876:   (0.09087599999999973, 12.0)
```

Trigonometric Functions

There are several built in trigonometric functions in Python. The function names and purpose are listed in table below.

Function	Description
sin(x)	Returns the sine of x radians.
cos(x)	Retums the cosine of x radians.
tan(x)	Returns the tangent of x radians.
asin(x)	Returns the arc sine of x, in radians.
acos(x)	Returns the arc cosine of x, in radians.
atan(x)	Returns the arc tangent of x, in radians.
atan2(y,x)	Returns atan(y/x), in radians.
hynot(xy)	Returns the Euclidean form, sqrt(x*x+yty).
degrees(x)	Converts angle x from radians to degrees.
radians(x)	Converts angle x degrees to radians.

Example

```
Import math
print("Sin(90), math.sin(90))
print ("Cos (90), math.cos (90))
print("Tan (90), math.tan (90))
print ("asin(1), math.asin(1))
print ("acos (1),math.acos(1))
 print("atan (1),nath.atan(1))
print ("atan2 (3,2), math.atan2(3,2))
print("Hypotenuse of 3 and 4, math.hypot (3,4))
print("Degrees of 90: math degrees (90))
("Radians of 90,math.radians (90))
```

Output

```
Sin(90) :        0.893996663601
Cos(90):         -0.448073616129
Tan (90):         asin (1): 1.57079632679
acos(1):         0.0
atan (1):        0.785398163397
atan2 (3,2):     0.982793723247
Hypotenuse of 3 and 4:          5.0
Degrees of 90:          5156.62015618
Radians of 90:          1.57079632679
```

Random Number Functions

There are several random number functions supported by Python. Random numbers have applications in research, games, cryptography, simulation and other applications. Table below shows the commonly used random number functions in Python. For using the random number functions, we **need to import the module random** because all these functions reside in the random module.

Function	Description
choice(sequence)	Returns a random value from a sequence like list, tuple, string etc.
shuffle(list)	Shuffles the items randomly in a list. Returns none.
random()	Returns a random floating point number which lies between 0 and 1.
randrange(start, stop, step)	Returns a randomly selected number from a range where start shows the starting of the range, stop shows the end of the range and step decides the number to be added to decide a random number.
seed(x)	Gives the starting value for generating a random number. Returns none. This function is called before calling any other random module function.
uniform(x,y)	Generates a random floating-point number n such that n>x and n<y.

Example

```
import random
s='abcde'
print("choice (abcde)", random.choice (s))
list = [10,2,3,1,3,19]
print("random.seed (20):", random.seed(20))
print ("random():", random.random())
print ("shuffle (list):", random.shuffle(list))
print ("randrange (2,10,1)", random, randrange (2,10,1))
print("uniform (2,3):", random. uniform (2,3))
```

Output

```
choice (abcde):          b
```

shuffle (list): None
random.seed (20): None
random(): 0.905639676175
uniform (2,3): 2.68625415703
randrange(2,10,1): 8

2 STRINGS

Strings in Python 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 ending at-1.The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Example

```
str = "Welcome to Python Programming
print (str)
print (str[0])
print (str[11:17])
print (str[11:])
print (str* 2)
```

Output

```
Welcome to Python Programming
W
Python
Python Programming
Welcome to Python Programming Welcome to Python Programming
```

Escape Characters

An escape character is a character that gets interpreted when placed in single or double quotes. They are represented using backslash notation. These characters are non printable. The following table shows the most commonly used escape characters and their description.

Escape Characters	Description
\a	Bell or alert
\b	Backspace
\n	Newline

\s	Space
\t	Tab
\t	Vertical Tab

String Formatting Operator

This operator is unique to strings and is similar to the formatting operations of the printf() function of the C programming language. The following table shows various format symbols and their conversion.

Format Symbols	**Conversion**
%c	Character
%s	String
%i	Signed Decimal Integer
%d	Signed Decimal Integer
%u	Unsigned Decimal Integer
%o	Octal Integer
%x	Hexadecimal Integer (lowercase letters)
%X	Hexadecimal Integer (uppercase letters)
%e	Exponent notation with lowercase letter ‘e’
%E	Exponent notation with uppercase letter ‘E’
%f	Floating point real number

String Formatting Functions

Python includes a large number of built-in functions to manipulate strings.

1. **len(string)** - Returns the length of the string.

Example Program

```
a= “Learning Python is fun!"
print (len(a))
```

Output: Learning Python is fun!

2. lower() - Returns a copy of the string in which all uppercase alphabets in a string are converted to lowercase alphabets.

Example Program

s= “Learning Python is fun!"

print (s.lower())

Output: learning python is fun!

3. upper() - Returns a copy of the string in which all lowercase alphabets in a string are converted to uppercase alphabets.

Example Program

s= ‘Learning Python is fun!’

print (s.upper())

Output: LEARNING PYTHON IS FUN!

4. swapcase() - Returns a copy of the string in which the case of all the alphabets are swapped. ie. all the lowercase alphabets are converted to uppercase and vice versa.

Example Program

s= “LEARNing python is fun!”

print (s.swapcase())

Output: learnING PYTHON IS FUN!

5. capitalize() - Returns a copy of the string with only its first character capitalized.

Example Program

s= ‘learning Python is fun!'

print (s.capitalize())

Output: Learning python is fun!

6. title() - Returns a copy of the string in which first character of all the words are capitalized.

Example Program

s='learning Python is fun!’

print (s.title())

Output: Learning Python Is Fun!

7. **lstrip()** - Returns a copy of the string in which all the characters have been stripped(removed) from the beginning. The default character is whitespaces.

Example Program

s= ' learning Python is fun!'

print (e.lstrip())

```
s= '******learning python is fun!'
print (s.1strip(***)))
```

Output: learning Python is fun!

learning Python is fun!

8. rstrip() - Returns a copy of the string in which all the characters have been stripped(removed) from the end. The default character is whitespaces.

Example Program

```
s='learning Python is fun!'
print (s.rstrip ())
s='learning Python is fun!***
print (s.rstrip (***))
```

Output: learning Python is fun!

learning Python is fun!

9. strip() - Returns a copy of the string in which all the characters have been stripped(removed) from the beginning and end. It performs both 1strip() and rstrip(). The default character is whitespaces.

Example Program

```
s=' learning Python is fun! '
print (s.strip())
s=*******learning Python is fun!******
print (s.strip (***))
```

Output: learning Python in fun!

learning Python is fun!

10. max(str) - Returns the maximum alphabetical character from string str.

Example Program

```
s='learning Python is fun!
print("Maximum character is: max(a))
```

Output: Maximum character is: y

11. min(str) - Returns the minimum alphabetical character from string str.

Example Program

```
S='learning –Python- is- fun'
print ("Minimum character is : min(s))
```

Output: Minimum character is: -

12. replace(old, new max) - Returns a copy of the string with all the occurrences substring old is replaced by new. The max is optional and if it is specified, the first occurrences specified in max are replaced.

Example Program

```
s="This is very new. This is good"
print (s.replace('is', 'was'))
```

Output

```
Thwas was very new. Thwas was good
```

13. center(width, fillchar) - Returns a string centered in a length specified in the width variable. Padding is done using the character specified in the fillchar. Defaul padding is space.

Example Program

```
s="This is Python Programming"
print (s.center (30, '*'))
print (s.center (30))
```

Output

```
**This is Python Programming**
   This is Python Programming
```

14. ljust(width,fillchar) - Returns a string left-justified in a length specified in the width variable. Padding is done using the character specified in the fill char. Default padding is space.

Example Program

```
s= "This is Python programming"
print(s.ljust(30, '*'))
print(s.ljust(30))
```

Output: This is Python programming****

This is Python programming

15. rjust(width,fillchar) - Returns a string right-justified in a length specified in the width variable. Padding is done using the character specified in the fill char.

Example Program

```
s= "This is Python programming"
print(s.rjust(30, '*'))
```

```
print(s.rjust(30))
```

Output: ****This is Python programming

This is Python programming

16. count(str, beg=0, end) – Returns the number of occurrence of str in the range beg and end.

Example Program

```
s= “This is Python Programming”
print(s.count(‘i’, 0, 10))
```

Output: 2

17. find(str, beg, end) – Returns the index of str if str occurs in the range beg and end returns -1 if it is not found.

Example Program

```
s= “This is Python Programming”
print(s.find(‘thon’, 0, 25))
print(s.find(‘thy’))
```

Output: 10

-1

18. startswith(suffix, beg=0,end) - It returns True if the string begins with the specified suffix, otherwise return False.

Example Program

```
s="Python programming is fun”
print (s.startswith('is',10,21))
```

Output: False

19. endswith(suffix, beg,end) - It returns True if the string ends with the specified suffix, otherwise return False.

Example Program

```
s="Python programming is fun”
print (s.endswith('is',10,21))
```

Output: False

20. islower() - Returns True if string has at least 1 cased character and all cased chan are in lowercase and False otherwise.

Example Program

```
S= “Python Programming"
print (s.islower())
```

Output: False

21. isupper() - Returns True if string has at least one cased character and all ca characters are in uppercase and False otherwise.

Example Program

```
s="Python Programming"
print (a.isupper ())
```

Output: False

22. isspace() - Returns True if string contains only whitespace characters and False otherwise.

Example Program

```
s="Python Programming"
print (s.isspace())
s= “  “
print (s. isspace())
```

Output: False

True

3. LIST

List is an ordered sequence of items. It is one of the most used data type in Python and is very flexible. All the items in a list do not need to be of the same type. Items separated by commas are enclosed within brackets. To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type. The values stored in a list can be accessed using the slice operator ([ ]and [:]) with indices starting at 0 in the beginning of the list and ending with -1. The plus (+) sign is the list concatenation operator and the asterisk (*) is the repetition operator.

Example Program

```
first_list = [‘abcd’, 147, 2.43, ‘Tom'. 74.9]
print (first_list)
print (first_list [0])
```

print (first_list(1:3])

Output:

[‘abcd’, 147, 2.43, ‘Tom'. 74.9]

abcd

[147, 2.43]

We can update lists by using the slice on the left hand side of the assignment operator. Updates can be done on single or multiple elements in a list.

Eg: list = [‘abcd’, 147, 2.43, ‘Tom', 74.9]
print(list[2])
list [2]=500
print(list[2])

Output: 2.43
500

To remove an item from a list, there are two methods. We can use del statement or **remove()** method.

Eg: list = [‘abcd’, 147, 2.43, ‘Tom', 74.9]
del list[2]
print(list)

Output: [‘abcd’, 147,‘Tom', 74.9]

Built-in List Functions

1. **len(list)** - Gives the total length of the list.

eg: list=[‘abcd’, 147, 2.43, ‘Tom', 74.9]
print (len(list1))

Output: 5

2. **max(list)** - Returns item from the list with maximum value.

Eg: list1= [1200, 147, 2.43, 1.12]
print("Maximum value in the list ", max(list1))

Output: Maximum value in the list 1200

3. **min(list)** - Returns item from the list with minimum value.

Eg: list1=[1200, 147, 2.43, 1.12]

```
print("Minimum value in the list is", min (list1))
```

Output: Minimum value in the list is 1.12

4. **list(seq)** - Returns a tuple into a list.

Eg:
```
tuple = ('abcd', 147, 2.43, 'Tom')
print( list (tuple))
```

Output: ['abcd', 147, 2.43, 'Tom']

5. **map(Function, Sequence)** - One of the common things we do with list and other sequences is applying an operation to each item and collect the result.

Eg:
```
str=input ("Enter a list")
li=list (map (int, str.split()))
print (li)
```

Output: Enter a list (space separated):1 2 3 4

[1, 2, 3, 4]

In the above example, a string is read from the keyboard and each item is converted into int using map(Function, Sequence) function.

Built-in List Methods

1. **list.append(obj)** - This method appends an object obj passed to the existing list.

Eg:
```
list = ['abcd', 147, 2.43, 'Tom']
print("Old List before Append:", list)
list.append(100)
print("New List after Append:", list)
```

Output: old List before Append: ['abcd', 147, 2.43, 'Tom']

New List after Append: ['abcd', 147, 2.43, 'Tom', 100]

2. **list.count(obj)** - Returns how many times the object obj appears in a list.

Eg:
```
list = ['abcd', 147, 2.43, 'Tom' ,147, 200, 147]
print("The number of times 147 appears in list=", list.count (147))
```

Output: The number of times 147 appears in list=3

3. **list.remove(obj)** - Removes object obj from the list.

Eg:
```
list1= ['abcd', 147, 2.43, 'Tom']
list1.remove('Tom')
```

print (list1)

Output: ['abcd', 147, 2.43]

4. **list.index(obj)** - Returns index of the object obj if found, otherwise raise an exception indicating that value does not exist.

Eg: list1 = ['abcd', 147, 2.43, 'Tom']

print (list.index (2.43))

Output: 2

5. **list.extend(seq)** - Appends the contents in a sequence seq passed to a list.

Eg: list1=['abcd', 147, 2.43, 'Tom']

list2=['def', 100]

list1.extend(list2)

print (list1)

Output: ['abed', 147, 2.43, 'Tom', 'def', 100]

6. **list.reverse()** - Reverses objects in a list.

Eg: list1 = ['abcd', 147, 2.43, 'Tom']

List1.reverse()

print (list1)

Output: [Tom, 2.43, 147, 'abcd']

7. **list.insert(index,obj)** - Returns a list with object obj inserted at the given index.

Eg: list1=['abcd', 147, 2.43, 'Tom']

list1.insert(2,222)

print(list1)

Output: ['abcd', 147, 222, 2.43, 'Tom']

8. **list.pop([index])** - Removes or returns the last object obj from a list. We can even pop out any item in a list with the index specified.

Eg: list1 = ['abcd', 147, 2.43, 'Tom']

list1.pop(-1)

print("List after poping:",list1)

Output: ['abcd', 147, 2.43]

9. **list.clear()** - Removes all items from alist.

Eg: list1 = ['abcd', 147, 2.43, 'Tom']

```
print("List before clearing:",list1)
list1.clear()
print("List after clearing:",list1)
print(list1)
```

Output: List before clearing: ['abcd', 147, 2.43, 'Tom']
List after clearing: [ ]

Using List as Stacks

The list can be used as stack (Last In First Out). Stack is a data structure where the last element added is the first element retrieved. To add an item to the top of the stack, use append(). To retrieve an item from the top of the stack, use pop().

Eg:
```
stack= [10, 20, 30, 40, 50]
stack.append(60)
print(stack)
stack.pop()
print(stack)
```

Output: [10, 20, 30, 40, 50, 60]
[10, 20, 30, 40, 50]

Using List as Queues

It is also possible to use a list as a queue, where the first element added is the first element retrieved. Queues are Last In First OUT (LIFO) data structure. But lists are not efficient for this purpose.

To implement a queue, Python provides a module called **collections** in which a method called **deque** is designed to have fast appends and pops from both ends.

Eg:
```
queue= deque (["apple", "orange", "pear"])
queue.append("cherry")
queue.append("grape")
queue.popleft()
```

Output: deque(['pear, cherry', 'grapes', 'cherry'])

4. TUPLE

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. The main differences between lists and tuples are lists are enclosed in square brackets ([ ]) and their elements and size can be changed, while taples are enclosed in parentheses (()) and cannot be updated. Tuples can be considered as read-only lists.

Example Program

```
t= (‘abcd', 147, 2.43, ‘Tom’, 74.9)
print (t)
print (t[0])
print (t [1:3])
print (t [2:))
```

Output

```
('abcd', 147, 2.43, Tom', 74.9)
abcd
(147, 2.43)
(2.43, "Tom", 74.9)
```

To delete an entire tuple we can use the **del** statement. Example **del tuple**. It possible to remove individual items from a tuple. However it is possible to create tuple which contain mutable objects, such as lists.

Built-in Tuple Functions

1. **len(tuple)** - Gives the total length of the tuple.

Eg:
```
tuplel = ('abcd', 147, 2.43, 'Tom')
print (len(tuple1))
```
Output: 4

2. **max(tuple)** - Returns item from the tuple with maximum value.

Eg:
```
tuple1= (1200, 147, 2 43, 1.12)
print ("Maximum value in tuplel”, max(tuple1))
```
Output: Maximum value in tuple 1200

3. **min(tuple)** - Returns item from the tuple with minimum value.

Eg:
```
tuplel= (1200, 147, 2.43. 1.12)
print ("Minimum value in tuplel is", min(tuplel))
```

Output: Minimum value in tuple is 1.12.

4. **tuple(seq)** - Returns a list into a tuple.

Eg: list = ['abcd', 147, 2.43, 'Tom']

print(tuple (list))

Output: ('abcd', 147, 2.43, 'Tom')

5. **SET**

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces {}. It can have any number of items and they may be of different types (integer, float, tuple, string etc.). Items in a set are not ordered. Since they are unordered we cannot access or change an element of set using indexing or slicing. We can perform set operations like union, intersection, difference on two sets. Set have unique values. They eliminate duplicates. The slicing operator [] does not work with sets. An empty set is created by the function set().

Eg: s1={1,2,3}

print (s1)

s2={1, 3,2.4, 'apple', 'Tom'}

print (s2)

Output: {1, 2, 3}

{1,3, 2.4, 'apple', 'Tom'}

Built-in Set Functions

1. **len(set)** - Returns the length or total number of items in a set.

Eg: set1= {'abcd', 147, 2.43. "Tom"}

print (len(set1))

Output: 4

2. **max(set)** - Returns item from the set with maximum value.

Eg: setl= {1200, 147, 2.43, 1.12}

print ("Maximum value is: ",max(set1))

Output: Maximum value is: 1200

3. **min(set)** - Returns item from the set with minimum value.

Eg: set1 = {1200, 147, 2.43.1.12)

```
print ("Minimum value is: ", min(set1))
```

Output: Minimum value is: 1.12

4. **sum(set)** - Returns the sum of all items in the set.

```
Eg:  setl={147, 2.43}
     print ("Sum of elements in set1 is", sum(set1))
```

Output: Sum of elements in set1 is 149.43

5. **sorted(set)** - Returns a new sorted list. The set does not sort itself.

```
Eg:  set1={213, 100, 289, 40, 23, 1, 1000}
      set2= sorted (set1)
     print(" elements before sorting:", set1)
     print(" elements after sorting:", set2)
```

Output: elements before sorting: {1, 100, 299, 1000, 40, 213, 23}

elements after sorting: {1, 23, 40, 100, 213, 289, 1000}

6. **any(set)** - Returns True, if the set contains at least one item, False otherwise.

```
Eg:  set1 = set ()
     set2={1,2,3,4}
     print (any (set1))
     print (any (set2))
```

Output: False

True

Built-in Set Methods

1. **setadd(obj)** - Adds an element obj to a set.

```
Eg:  set1={3,8,2,6}
     set1.add(9)
     print ("Set after addition:", set1)
```

Output: Set before addition: {8, 2,9, 3, 6}

2. **set.remove(obj)** - Removes an element obj from the set. Raises KeyError if the set is empty.

```
Eg:  set1={3,8,2,6}
     set1.remove(8)
```

print ("Set after deletion:", setl)

Output: Set after deletion: {2, 1, 6}

3. **set.discard(obj)** - Removes an element obj from the set. Nothing happens if the element to be deleted is not in the set.

Eg: set1={3, 8, 2, 6}

set1.discard(8)

print("Set after discard:", set1)

Output: Set after discard: {2, 3, 6}

4. **set.pop()** - Removes and returns an arbitrary set element. Raises KeyError if the set is empty.

Eg: set1={3,8,2,6}

set1.pop()

print ("Set after poping:", set1)

Output: Set after poping: {2, 3, 6}

5. **sell.union(set2)** - Returns the union of two sets as a new set.

Eg: set1={3,8,2,6}

set2={4,2,1,9}

set3=set1.union (set2)

print ("Union:", set3)

Output: Union: {1, 2, 3, 4, 6, 8, 9}

6. **setl.update(set2)** - Update a set with the union of itself and others. The result will be stored in set1.

Eg: set1={3,8,2,6}

set2={4,2,1,9}

setl.update(set2)

print ("Update Method:", set1)

Output: Update Method: {1, 2, 3, 4, 6, 8, 9}

7. **setl.intersection (set2)** - Returns the intersection of two sets as a new set.

Eg: set1={3,8,2,6}

set2={4,2,1,9}

set3= set1.intersection (set2)

print ("Intersection: ", set3)

Output: Intersection: {2}

8. **set1.difference(set2) -** Returns the difference of two or more sets into a new set.

Eg: set1={3,8,2,6}

set2={4,2,1,9}

set3= setl.difference (set2)

print("Difference:", set3)

Output: Difference: {8, 3, 6}

Frozenset

Frozenset is a new class that has the characteristics of a set, but its elements cannot changed once assigned. While tuples are immutable lists, frozensets are immutable sets.

Frozensets can be created using the function frozenset(). This datatype supports methods like difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric_dillerence() and union(). Being immutable it does not have methods like add(), remove(), update(), difference update(),intersection_update(),symmetric difference_update() etc

6. DICTIONARY

Dictionary is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. We must know the key to retrieve the value. In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type. Keys are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object. Dictionaries are sometimes found in other languages as "associative memories or associative arrays".

Dictionaries are enclosed by curly braces ({}) and values can be assigned and accessed using square braces ([ ]).

Example Program

```
dict={'name': 'john', 'code':6734,  'dept':'sales'}
print (dict)
print (dict.keys())
print (dict.values())
```

Output: {'name': 'john', 'code':6734, 'dept':'sales'}

{'name', 'code', 'dept'}

{ 'john',6734,'sales'}

We can delete the entire dictionary elements or individual elements in a dictionary. We can use **del** statement to delete the dictionary completely. To remove entire elements dictionary, we can use the **clear()** method.

Properties of Dictionary Keys

1. More than one entry per key is not allowed.ie, no duplicate key is allowed. When duplicate keys are encountered during assignment, the last assignment is taken.
2. Keys are immutable. This means keys can be numbers, strings or tuple. But it does not permit mutable objects like lists.

Built-in Dictionary Functions

1. **len(dict)** - Gives the length of the dictionary.

Eg: dict1= {'Name': 'Tom', 'Age' :20, 'Height':160}

print ("Length of Dictionary=", len (dict1))

Output: Length of Dictionary= 3

2. **str(dict)** - Produces a printable string representation of the dictionary.

Eg: dict1= {'Name': 'Tom', 'Age' :20, 'Height':160}

print ("Representation of Dictionary", str(dict1))

Output: Representation of Dictionary ('Age': 20, 'Name': 'Tom', 'Height': 160)

3. **type(variable)** - The method type() returns the type of the passed variable. If passed variable is dictionary then it would return a dictionary type. This function can be applied to any variable type like number, string, list, tuple etc.

Eg: dict1= { 'Name': 'Tom', 'Age':20, 'Height':160}

print (type (dict1))

s="abcde"

print(type(s))

Output: <class 'dict'>

<class 'str'>

Built-in Dictionary Methods

1. **dict.clear()** - Removes all elements of dictionary dict.

Eg: dict1= {‘Name’:’Tom', 'Age’:20, ‘Height’:160}

dict1.clear()

print (dicti)

Output: { }

2. **dict.copy() -** Returns a copy of the dictionary dict.

Eg: dict1= {'Name' :’Tom', 'Age’:20, ‘Height’:160}

dict2=dict1.copy()

print (dict2)

Output: {‘Age’: 20, ‘Name’: ‘Tom’, ‘Height’: 160}

3. **dict.keys()** - Returns a list of keys in dictionary dict.

Eg: dict1= {'Name' :’Tom', 'Age’:20, ‘Height’:160}

dict1.keys()

print (dict1)

Output: ['Age', 'Name', 'Height']

4. **dict.values()** - This method returns list of all values available in a dictionary.

Eg: dict1= {'Name' :’Tom', 'Age’:20, ‘Height’:160}

print ("Values in Dictionary,dicti.values())

Output: Values in Dictionary: [120, ‘Tom’, 160]

5. **dict.items()** - Returns a list of dictionary dict's(key,value) tuple pairs.

Eg: dict1= {'Name' :’Tom', 'Age’:20, ‘Height’:160}

print (dict1.items())

Output: [('Age', 20), ('Name', 'Tom'), ('Height', 160)]

6. **dict1.update(dict2)** - The dictionary dict2's key-value pair will be updated in dictionary dict1.

Eg: dict1= {'Name' :’Tom', 'Age’:20, ‘Height’:160}

dict2= {'Weight’: 60}

dict1.update(dict2)

print (dict1)

Output: dict1= {'Name' :’Tom', 'Age’:20, ‘Weight’=60, ‘Height’:160}

MUTABLE AND IMMUTABLE OBJECTS

Objects whose value can change are said to be mutable and objects whose value is unchangeable once they are created are called immutable. An object's mutability is determined by its type. For instance, numbers, strings and tuples are immutable, while lists, sets and dictionaries are mutable.

DATA TYPE CONVERSION

We may need to perform conversions between the built-in types. To convert between different data types, use the type name as a function. There are several built-in function perform conversion from one data type to another. These functions return a new representing the converted.

Function	Description
int (x)	Converts x to an integer
long(x)	Converts x to a long integer.
float(x)	Converts x to a floating point number.
str(x)	Converts x to a string representation.
tuplet(s)	Converts to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character
hex(x)	Converts an integer to an hexadecimal string.
oct(x)	Converts an integer to an octal string.