PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Mohan Kumar AV, Sindhu Pai
Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

List

Mohan Kumar AV, Sindhu Pai
Department of Computer Science and Engineering

List

A **List** is a Collection. List is an ordered sequence of items. Values in the list are called elements / items.

A collection allows us to put many things / values under a single name called **“variable”.**

List is a linear data structure where elements have linear ordering.

List

Creation of list

- **List** items are surrounded by square brackets and the elements in the list are separated by commas.

```
politicians=['modi', 'rahul', 'mamta', 'kejriwal']
```

- A list element can be any Python object - even **another list**.

```
politicians=['modi', 'yediyurappa','devegowda',['parikar', 'swaraj', 'jately']]
```

- A list can be empty.

```
lst=[ ]
```

List

List characteristics:

- Elements in the list can be heterogeneous in nature. Items in the lists can be of different data types.

  Ex:

  ```
  hetero_list=[1,"rahul",3.5,{1,2,3}]
  ```

- Homogeneous list

  Ex:

  ```
  sports=["tendulkar","bolt"," federer","messi"]
  ```

- List elements can be accessed by index.

List

List characteristics:

- Lists are mutable.
- List is iterable - is eager and not lazy.
- Assignment of one list to another causes both to refer to the same list.
- List can be sliced. This creates a new (sub)list.

List

- Elements are accessed using indexing operation or by subscripting.

```
>>> numbers = [ 12,78,33,32.7,11.9,83,78]
>>> print ( numbers [1])
    78
```

Note: List index always starts with 0 , also called as zero based indexing.

List

- Lists are mutable, as list can grow or shrink .
    - We *can* change an element of a list.

Ex:

```
>>> numbers=[55,88,45,12]
>>> numbers[0]=10 # index operation is used.
>>> numbers
    [10, 88, 45, 12]
```

List

- List is iterable - is eager and not lazy.

Ex: (i)
```
numbers=[55,88,45,12]
for i in numbers:
        print(i, end =' ')
```

Ex: (ii)
```
number=[10,20,30,40,50]
i=0
while(i<len(number)):
        print(number[i],end=' ')
        i=i+1
```

- List can be nested. We can have list of lists.

Ex: (i)
```
numbers=[55,20,[63,72,33]]
for i in numbers:
        print(i, end =' ')
```

Ex: (ii)
```
number=[55,20,[63,72,33]
i=0
while(i<len(number)):
        print(number[i],end=' ')
        i=i+1
```

List

- Assignment of one list to another causes both to refer to the same list.

Ex:

```
>>> list1=[12,44,55,89,11,24]
>>> list2=list1
>>> print(id(list1))
    2894590353408
>>> print(id(list2))
    2894590353408
```

Note: In Python, the id() function is a built-in function that returns the unique identifier of an object.

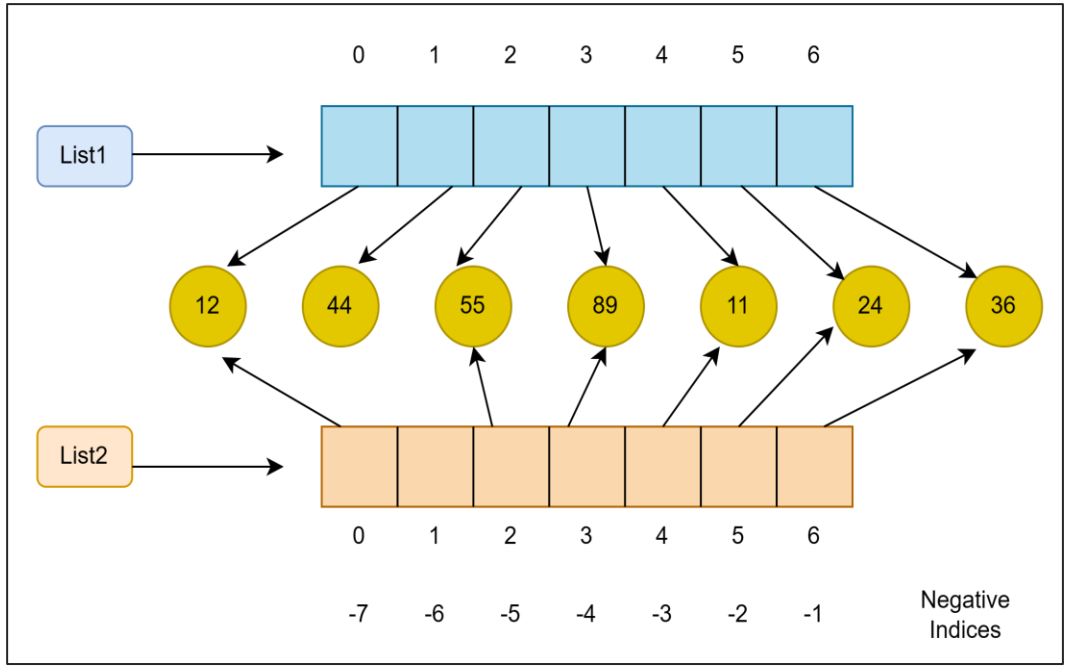
List

Given lst1 = [12,44,55,89,11,24]

```
>>> lst2 = lst1[::] # creates a copy of lst1. Not same as lst2 = lst1

>>> print(id(lst1))
    2894635511936

>>> print(id(lst2)) #These two are different values
    2894635298304
```



List

- List can be sliced. This creates a new (sub)list.

Ex:

```
>>> lst = [9, 41, 12, 3, 74, 15]
>>> lst[1:3]
    [41,12]
>>> lst[:4]
    [9, 41, 12, 3]
>>> lst[3:]
    [3, 74, 15]
>>> lst[:]
    [9, 41, 12, 3, 74, 15]
```

```
>>> lst1=[10,20,[30,40,50]]
>>> lst1[2][1]
    40
```

PES UNIVERSITY

Built in Functions

There are a number of functions built into Python that take lists as parameters.

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
```

List

- **Concatenation**

We can create a new list by adding two existing lists together.

Ex:

```
>>> list1 = [87,46,34,40,12]
>>> list2 =[23,32,86,32,11]
>>> list1 + list2
    #concatenates two lists
    [87,46,34,40,12, 23,32,86,32,11]
```

List

- **Repetition**

Operation allows multiplying the list n times.

Ex:

```
>>> list1 = [23,32,86,32,11]
>>> list1 * 2
    [23,32,86,32,11, 23,32,86,32,11]
```

PES UNIVERSITY

Membership Operator:

in and not in

It returns true if a particular item exists in the list otherwise false.

Ex:

```
>>> list1 = [12, 'Sun',39, 5,'Wed', 'Thus'] # heterogeneous list.
>>> 'Wed' in list1
    True
```

List

The not in operator returns True if the element is not present in the tuple, else it returns False.

Ex:

```
>>> list1 = [12, 'Sun',39, 5,'Wed', 'Thus'] # heterogeneous list.
>>> 'ruby' not in list1
        True
```

PES UNIVERSITY

- **Comparison**

We may at times need to compare data items in the two lists to perform certain operations by using == operator.

Ex:

```
>>> list1 = [10,2.2,(22,33,43]
>>> list2=[2,3,4]
>>> list1==list2
    False
>>> list1!=list2
    True
```

List

The operations can be performed on List :

- **append()**

Allows to add element at the end of list.

Ex:

```
>>> list1 = [10,20,30,40,50]
>>> list1.append(22)
>>> list1
    [10,20,30,40,50,22]
```

List

- **insert(pos,val)**

Allows to add an element at particular position in the list.

Ex:

```
>>> list1 = [10,20,30,40,50]
>>> list1.insert(3,55)
>>> list1
    [10,20,30,55,40,50]
```

List

- **extend()**

Adds the specified list elements (or any iterable) to the end of the current list.

Ex:

```
>>> list1 = [10,20,30,40,50]
>>> list1.extend([11,22,33,44,55])
 >>> list1
      [10,20,30,40,50,11,22,33,44,55]
```

List

- **pop() & remove()**

Allows to remove element from a list by using pop() or remove() functions.

One uses index value (pop),another uses value(remove) as reference to remove the element.

Ex:

```
>>> list1 = [10,20,30,40,50]
```

```
>>>list1.pop(2) # using pop()
>>> list1
[10,20,40,50]
```

```
>>> list1.remove(40) #using remove()
>>> list1
[10,20,30,50]
```

List

- **count(val)**

Returns number of occurrences of value.

Ex:

```
>>> list1 = [10,20,30,40,50]
>>> list1.count(20)
    1
```

List

- **copy()**

Return a shallow copy of a list, which returns a new list without modifying the original lists.

Ex:

```
>>> lis = [‘23’,’13’,’45’]
>>> new_list = lis.copy()
>>> print('Copied List:', new_list)
```

List

- **Index (val)**

Return first index of a value. Raises Value error if the value is not present.

Ex:

```
>>> list1 = [45,20,30,15,67]
>>> list1.index(20)
    1
```

List

- **Sorting**

Allows to arrange the elements of a list.

Ex:

```
>>> list1 = [10, 1, -2, 2, 9]
>>> list1.sort()
>>> list1
    [-2, 1, 2, 9, 10]
```

- **dir()**

Returns all properties and methods of the specified object, without the values.

Ex:

```
number = [12]
# returns valid attributes of the number list
print(dir(number))
```

List

PES UNIVERSITY

Use of for and while Loops for list

1. for loop

2. while loop

PES UNIVERSITY

1. List using for Loop

- The for loop in Python is used to iterate over a sequence or other iterable objects.
- Iterating over a sequence is called traversal.
- Loop continues until we reach the last item in the sequence
- The body of for loop is separated from the rest of the code using indentation.

List

Accessing Element	Output
a = [34,100,23,45,56,145] for i in a: print(i)	34 100 23 45 56 145
a = [34,100,23,45,56,145] for i in range(0,len(a),1): print(i)	0 1 2 3 4 5
a = [34,100,23,45,56,145] for i in range(0,len(a),1): print(a[i])	34 100 23 45 56 145

2. List using while loop:

- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.
- When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

List

Ex: Python code to find Sum of elements in a list.

```
a=[1,2,3,4,5]
i=0
s=0
while i<len(a):
        s=s+a[i]
        i=i+1
print(s)
```

Output:

15

THANK YOU

Mohan Kumar AV, Sindhu Pai

Team Python - 2022

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Mohan Kumar AV, Sindhu Pai
Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Tuple

Mohan Kumar AV, Sindhu Pai
Department of Computer Science and Engineering

Tuple

Tuple is a data structure containing zero or more elements.

Creation of tuple

- An empty tuple can be created using a constructor, tuple () or ().

Ex:

```
>>> t1= ()
>>> print(type(t1)) # <class 'tuple'>
>>> t2=tuple ()
>>> print(type(t2)) #<class 'tuple'>
```

- Tuple with two elements.

```
>>> t3= (1,2)
```

Tuple

- Tuple with a single element

Ex:

```
>>> t1= (1)
>>> print(type(t1)) # <class 'int'>
>>> t1= (1,)
>>> print(type(t1)) # <class 'tuple'>
>>> t3= (1,2)
```

Tuple

Tuple has the following attributes:

- The element of the tuple can be of any type.

(There is no restriction on the type of the element.)

Ex:

```
t = ('python',1,3.4) # heterogeneous
t1 = (11,22,33,44) #homogeneous
```

Tuple

- Each element of a tuple can be referred or accessed by an index or a subscript and index is an integer.

Ex:

```
#tuple1 of even elements from 0 to 20
>>> tuple1=(2,4,6,8,10,12,14,16,18,20)
>>> tuple1[0] # 2 , print first element of tuple1
>>> tuple1[3] # 8 , print the fourth element of tuple1
>>> tuple1[10] #returns index out of range error
>>> tuple1[1+4] # 12 , an expression resulting in an integer index
# also supports negative indexing.
>>> tuple1[-1] # 20 , returns first element from right
```

Tuple

- Tuples are immutable. Once created, we cannot change the number of elements.

Ex:

```
>>> t1= (1, 2, 3, 4)
>>> len(t1) # 4
>>> t1[0]=100
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tuple

- Tuple is an iterable.

Ex:

```
t1=(1,3,2,5,6)
for i in t1:
        print(i, end=' ')
```

- Tuple can be nested.

Ex:

```
>> t1=(1,2,(11,22))
```

Tuple

- Assignment of one tuple to other causes both to refer to the same tuple.

Ex:

```
>>> t1=(1,2,3,4)
>>> t2=t1
>>> id(t1)
2490371269400
>>> id(t2)
2490371269400
```

Tuple

- Tuples can be sliced that creates a new (subset of) tuple.

Ex:

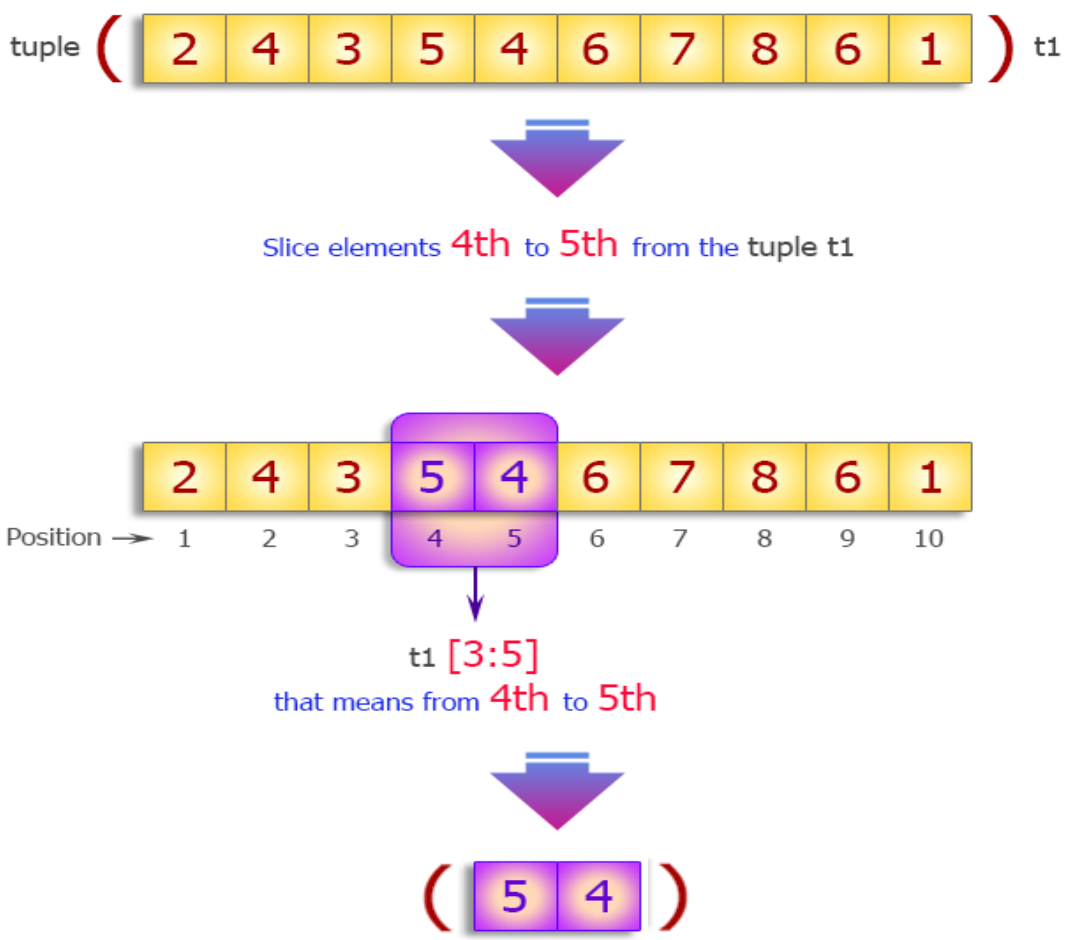
```
>>> t1
(1, 2, 3, 4)
>>> t1[2:3]
(3,)
```

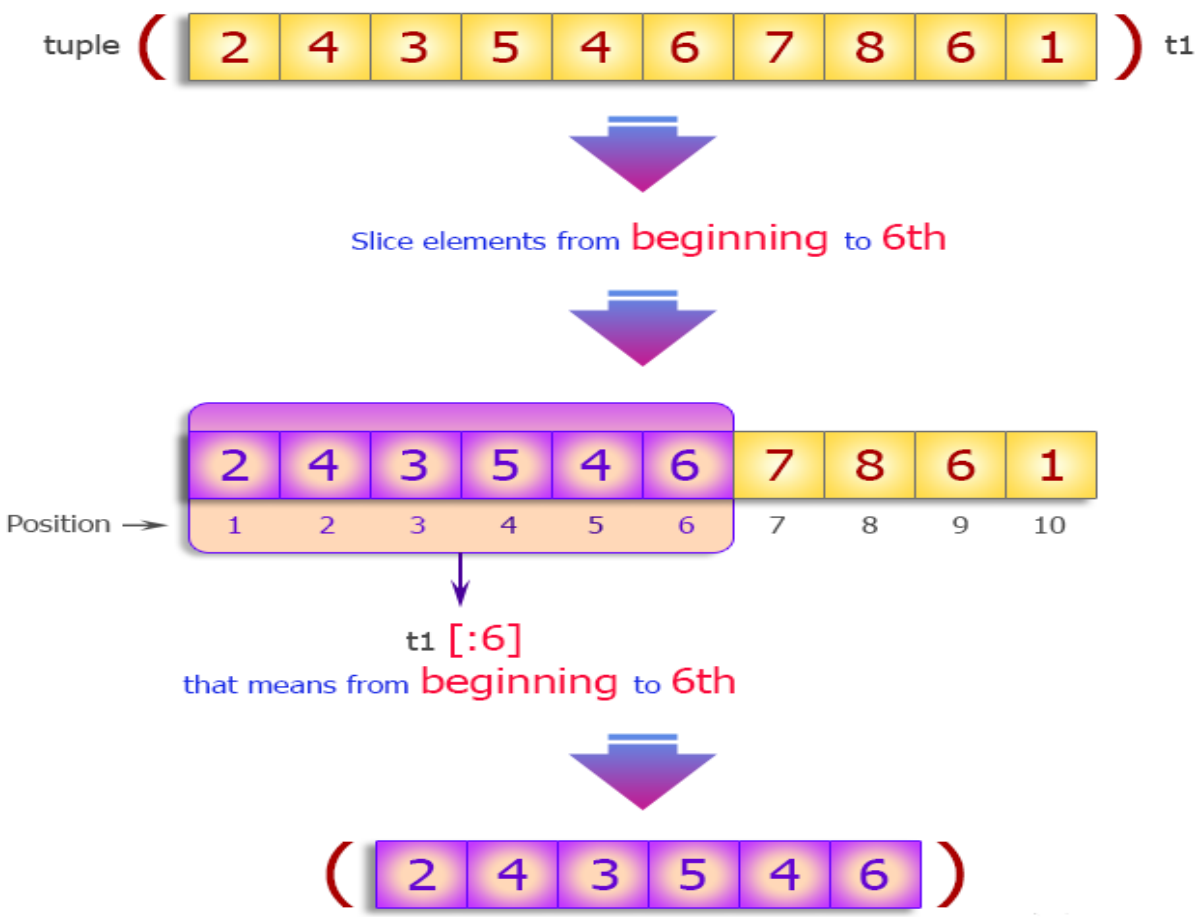
Note: If we use :: , it doesn't create a copy in tuple unlike list.

```
>>> t1=(1,2,3,4)
>>> t2=t1[::]
>>> print(id(t1))
    2940764604192
>>> print(id(t2))
    2940764604192
```

Tuple

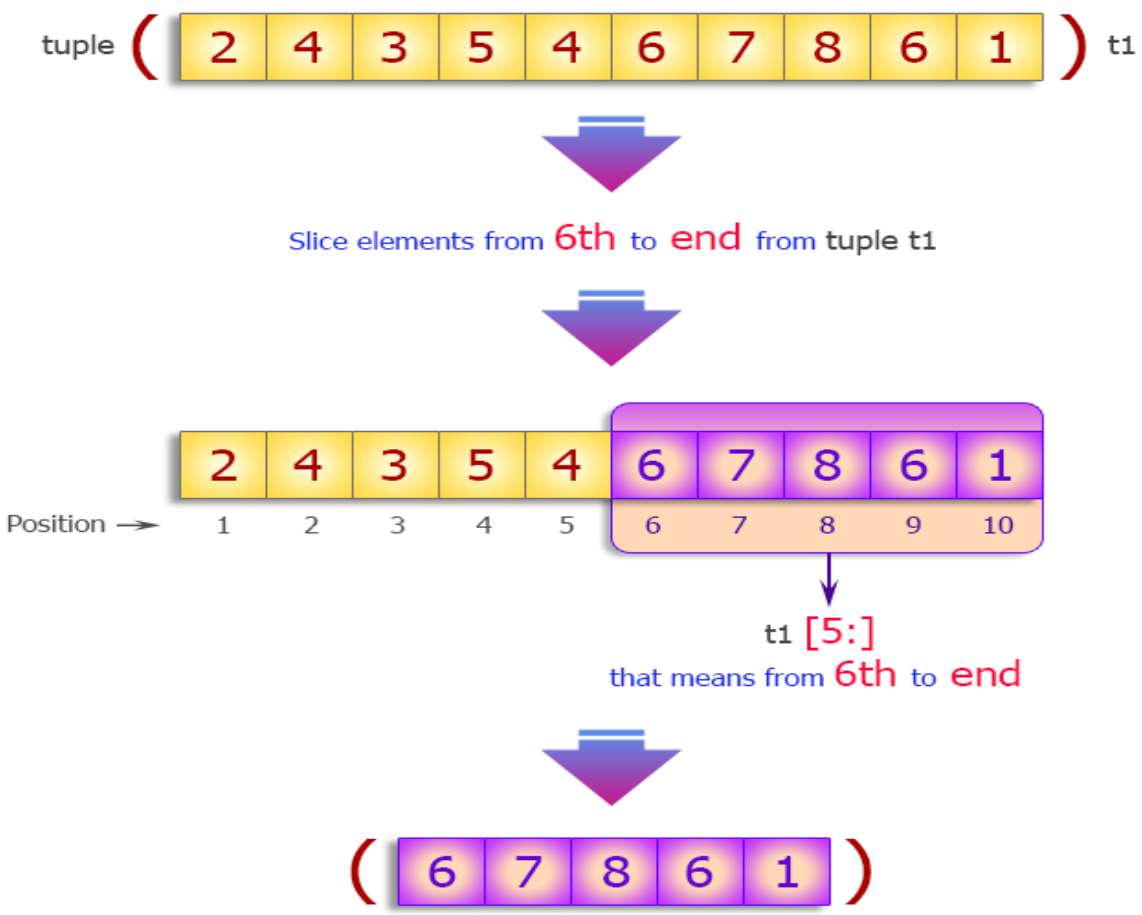
- **Slicing a tuple**

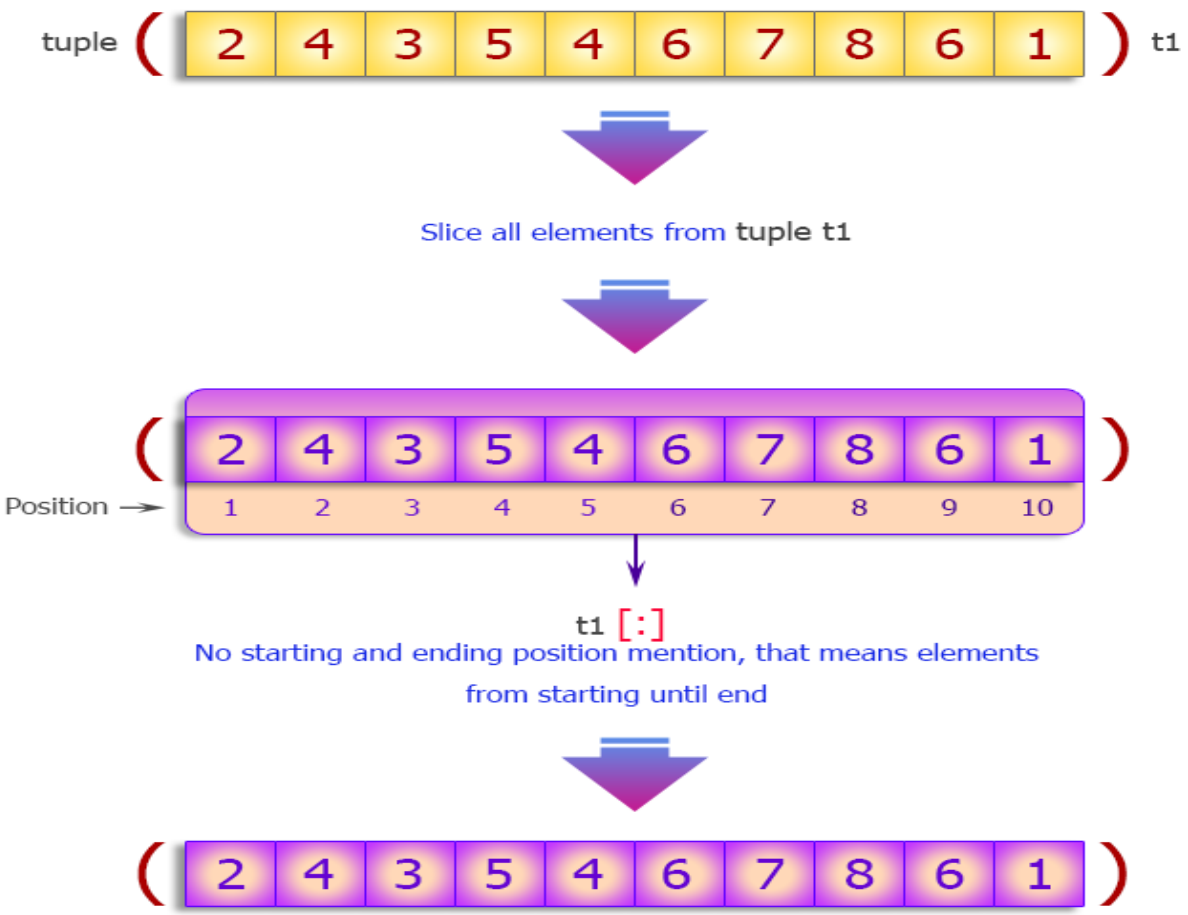




Tuple

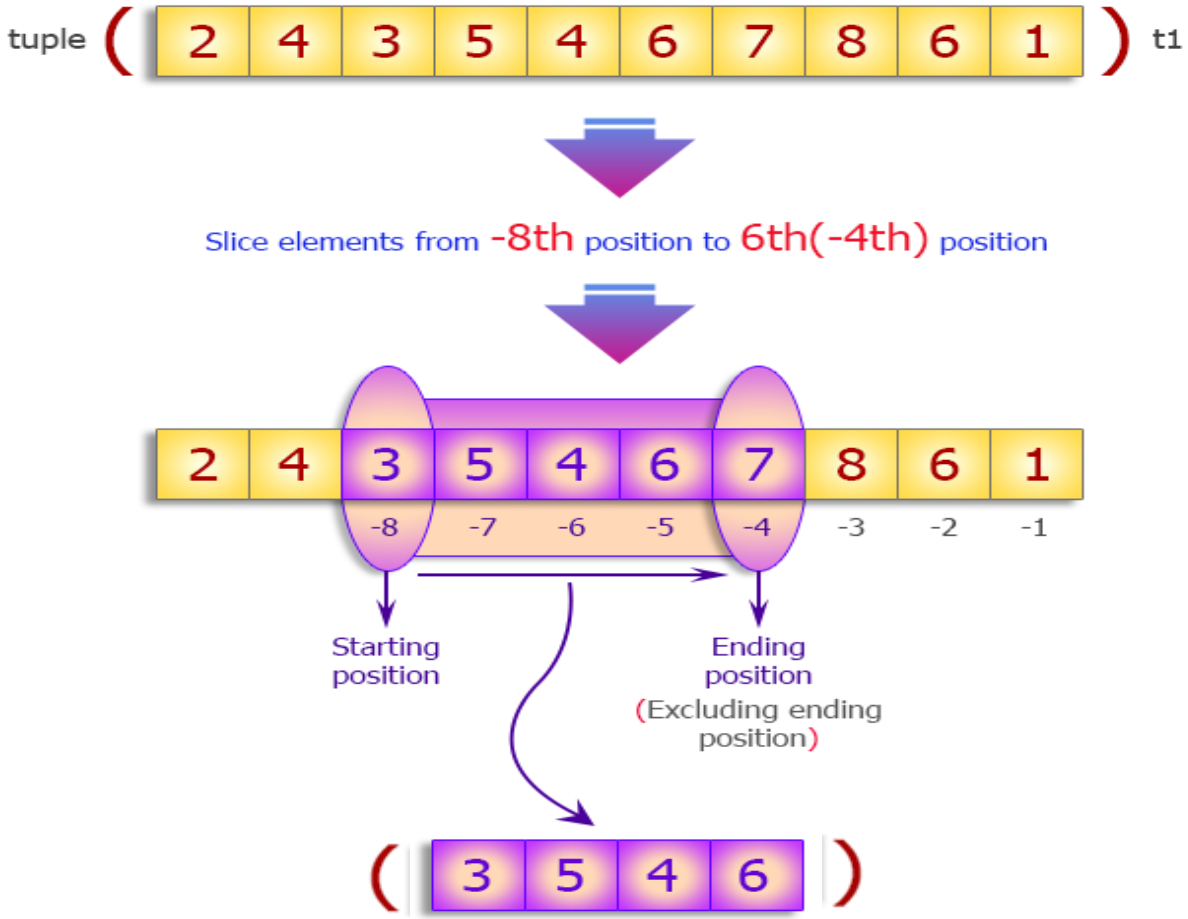
- **Slicing a tuple**

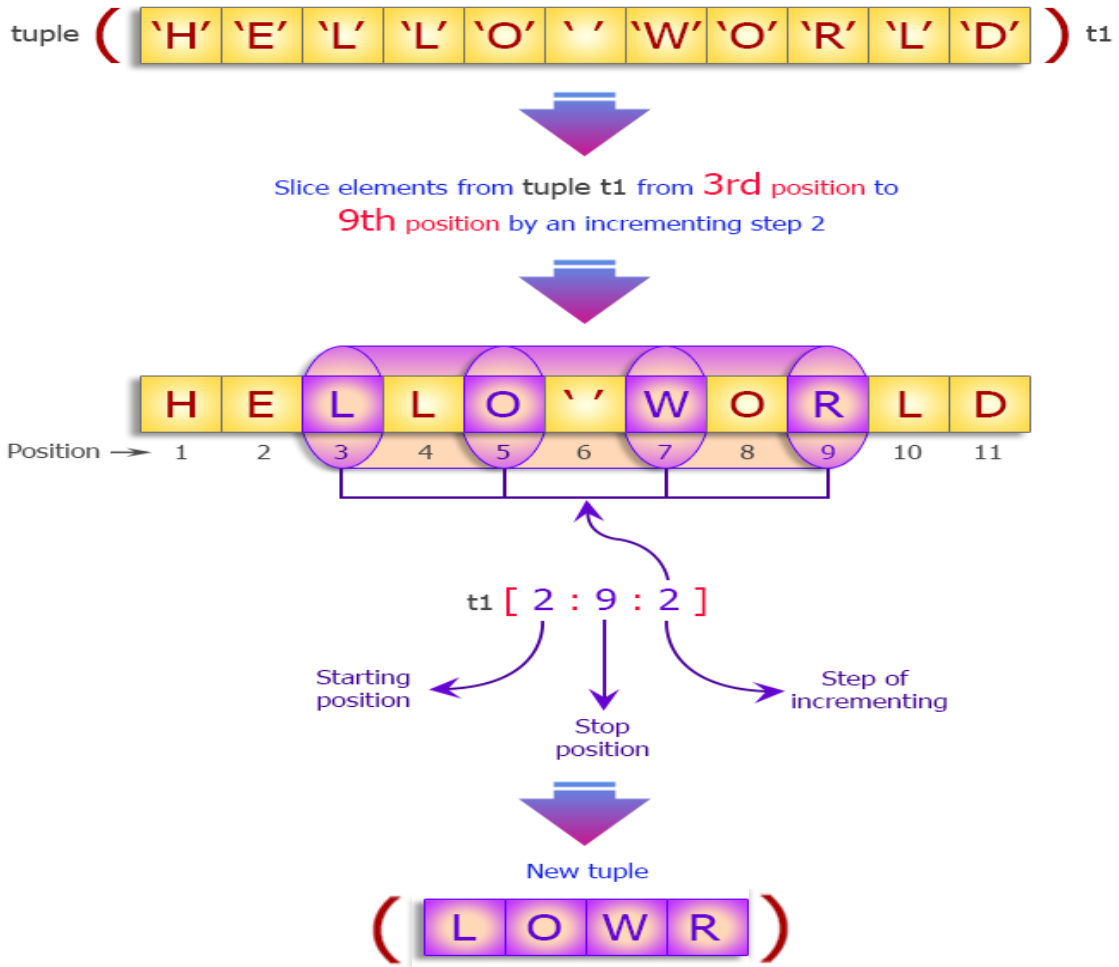




Tuple

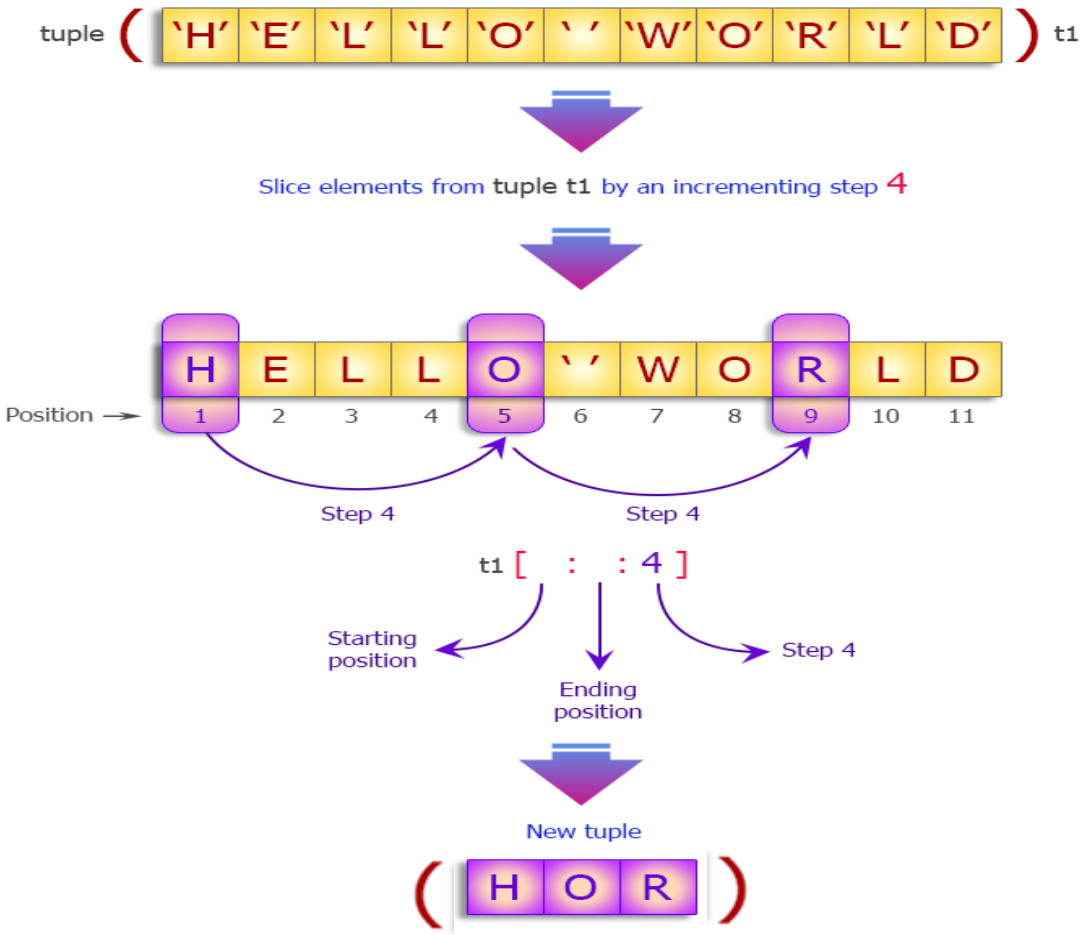
- **Slicing a tuple**

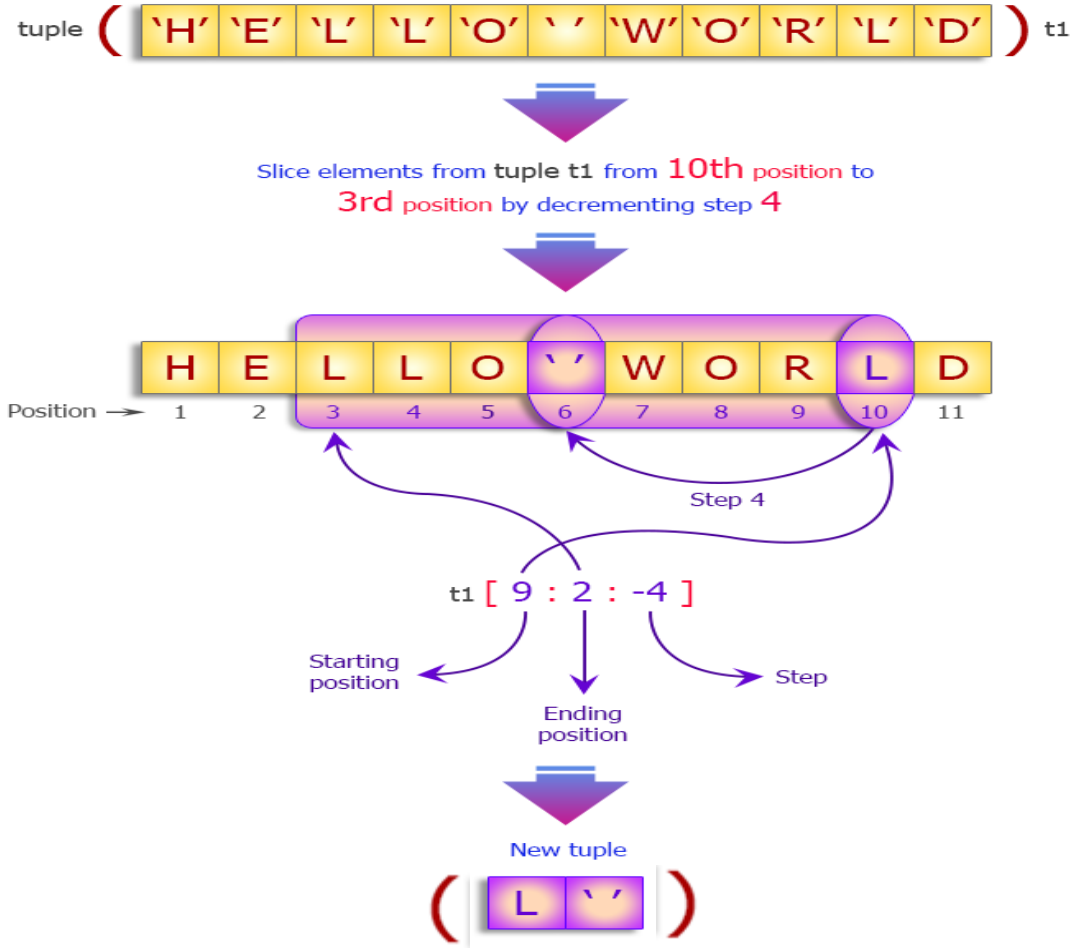




Tuple

- **Slicing a tuple**





Tuple

PES UNIVERSITY

Built in Functions

Method	Description	Example

len(tup)	Gives the total length of the tuple.	>>> tuple1 = (10,20,30,40,50)
		>>> len(tuple1)
		5

sorted()	Takes elements in the tuple and returns a new sorted list. It should be noted that, sorted () does not make any change to the original tuple	>>> tuple1 = (‘rama’,’shama’,’bhama’,’balarama’)
		>>> sorted(tuple1)
		[‘balarama’,bhama’,’rama’,’shama’]

Tuple

PES UNIVERSITY

Built in Functions

Method	Description	Example

min()	Returns the element from tuple with max value	>>> tuple1=(22,33,11,55,44,120)
		>>> min(tuple1)
		11

max()	Returns the element from tuple with min value	>>> tuple1=(22,33,11,55,44,120)
		>>> max(tuple1)
		120

sum()	Returns the sum of elements of the tuple.	>>> tuple1=(22,33,11,55,44,120)
		>>> sum(tuple1)
		285

Tuple

PES UNIVERSITY

Built in Functions

Method	Description	Example

tuple(seq)	Convert sequence into tuple	>>> tuple1 = tuple()
		>>> tuple1
		( )
		>>> tuple1 = tuple('a e I o u')#string
		>>> tuple1
		('a', 'e', 'i', 'o', 'u')
		>>> tuple2 = tuple([1,2,3]) #list
		>>> tuple2
		(1, 2, 3)
		>>> tuple3 = tuple(range(5))
		>>> tuple3
		(0, 1, 2, 3, 4)

Tuple

- Concatenation:

allows us to join two tuples by using operator '+'.

Ex:

```
>>> tuple1 = (11,33,55,77,99)
>>> tuple2 = (22,44,66,88,100)
>>> tuple1 + tuple2    #Concatenates two tuples
    (11, 33, 55, 77, 99, 22, 44, 66, 88 100)
```

Tuple

- Repetition:

The repetition operator enables the tuple elements to be repeated multiple times.

The repetition operator requires the first operand to be a tuple and the second operand to be an integer only.

Ex:

```
>>> t1=(1,2,3,4)
>>> t1*2
    (1, 2, 3, 4, 1, 2, 3, 4)
>>> t1 * t1
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'tuple
```

Tuple

PES UNIVERSITY

Membership Operator:

in and not in

It returns true if a particular item exists in the tuple otherwise false

Ex:

```
>>> tuple1 = (1,2.2,[22,33,43],”python”)
>>> “python” in tuple1
    True
```

Tuple

The not in operator returns True if the element is not present in the tuple, else it returns False.

Ex:

```
>>>  tuple1 = (1,2.2, [22,33,43],"python")
 >>> "PYTHON " not  in tuple1
        True
```

Tuple

PES UNIVERSITY

Functions :

Method	Description	Example

count()	Returns the count of the items.	>>> tuple1 = (10,20,30,10,40,10,50)
		>>> tuple1.count(10)
		3
		>>> tuple1.count(90)
		0

index()	Returns the index of the item specified	>>> tuple1 = (10,20,30,40,50)
		>>> tuple1.index(30)
		2
		>>> tuple1.index(90)
		ValueError: tuple.index(x): x not in tuple.

THANK YOU

Mohan Kumar AV, Sindhu Pai

Team Python - 2022

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Mohan Kumar AV, Sindhu Pai
Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Dictionary

Mohan Kumar AV, Sindhu Pai
Department of Computer Science and Engineering

Dictionary

A “bunch” of values, each with its own label.

Bag

passport

money

PES UNIVERSITY

Dictionary

- Dictionary is a data structure that organizes data into key and value pairs.

Dictionary Creation

```
>>> phonebook={} #Creation of empty Dictionary
>>> phonebook={"Johan":938477565} #Dictionary with one key-value pair
>>> phonebook={"Johan":938477565,"Jill":938547565} #Dictionary with two key-value pair
```

Dictionary

```
>>> eng2french={      1:’un’,
                      2:’deux’,
                      3:’trios’,
                      4:’quatre’,
                      5:’cinq }
```

To access values in the dictionary, we use the keys.

```
>>> print(end2french[2]) #will give you ‘deux’
```

#unordered

```
>>> d={'w':11,'a':33,'e':44}

>>> f= {'e':44,'w':11,'a':33}

>>> print(d==f)
```

True

Dictionary

PES UNIVERSITY

Characteristics:

- Dictionary is a data structure that organizes data into key and value pairs.

```
d={1:"one",2:"two",3:"three",4:"four"}
```

- In mathematical language, a dictionary represents a mapping from keys to values, so you can also say that each key “maps to” a value.
- Dictionary is mutable, associative data structure of variable length of key-value pairs.

Dictionary

- Each key is of any immutable type associated with a single value.

```
>>> d={1:"one",2:"two",3:"three",4:"four"}
>>> d1={[1,2]:"hello"}
        Traceback (most recent call last):
         File "<stdin>", line 1, in <module>
        TypeError: unhashable type: 'list'
```

- The values can be of any type .

```
>>> d={1:"one", 2:[23,33] , 3:"three", 4:"four"}
```

Dictionary

- If you assign a value to a key, then later in the same dictionary have the same key assigned to a new value, the previous value will be overwritten.

```
>>> d={1:"one",2:"two",3:"three"}
>>> d={1:"one",2:"two",3:"three",1:"five"}
>>> d
{1: 'five', 2: 'two', 3: 'three'}
```

Dictionary

- The items (key-value pair) in dictionary is unordered type, which means that the order isn't decided by the programmer but rather the interpreter.

- The ordering is based on a concept called "**Hashing**" .

- **Note**: hash() function in python is used to return a hashed integer value of the object we pass as a parameter into it.

Dictionary

PES UNIVERSITY

Common operation on Dictionary

- len()
- min()
- max()

❑ **Note:**

Does not support '+' and '*' operations.

Dictionary

Functions to play with dictionary.

- **get():** returns the value for a given key, if present.

```
>>> print(phonebook.get('Jill'))
    938547565
```

- **items(...)**

    D.items() -> a set-like object providing a view on D's items.

```
>>> phonebook.items()
    dict_items([('Johan', 938477565), ('Jill', 938547565)])
```

Dictionary

Functions to play with dictionary.

- **keys(...)**

  D.keys() -> a set-like object providing a view on D's keys.

```
>>> phonebook.keys()
    dict_keys(['Johan', 'Jill'])
```

Dictionary

Functions to play with dictionary.

- **pop(...)**

D.pop(key) -> v, remove specified key and return the corresponding value. If key is not found, otherwise KeyError is raised.

```
>>> phonebook.pop('Jill')
    938547565
```

Dictionary

PES UNIVERSITY

Functions to play with dictionary.

- **popitem(...)**

D.popitem() -> (k, v), remove and return some (key, value) pair as a 2-tuple; but raise KeyError if D is empty.

```
>>> person = {'name': 'Phill', 'age': 22, 'salary': 3500.0}
>>> result = person.popitem()
>>> print('Return Value = ', result)
    Return Value =  ('salary', 3500.0)
>>> print('person = ', person)
    person =  {'name': 'Phill', 'age': 22}
```

Functions to play with dictionary.

- **setdefault(...)**

D.setdefault(key,value) -> if the key is in the dictionary, returns its value. If the key is not present, insert key with a value of dictionary and return dictionary.

```
>>> person = {'name': 'Phill', 'age': 22}
>>> age = person.setdefault('age’)
>>> print('person = ',person)
>>> print('Age = ',age)
```

Dictionary

Functions to play with dictionary.

- **update(...)**

  D.update() -> It will update to the dictionary.

```
>>> marks = {'Physics':67, 'Maths':87}
>>> internal_marks = {'Practical':48}
>>> marks.update(internal_marks)
>>> print(marks)
    {'Physics': 67, 'Maths': 87, 'Practical': 48}
```

Dictionary

Functions to play with dictionary.

- **values(...)**

D.values() -> returns a view object that displays a list of all the values in the dictionary.

```
>>> marks = {'Physics':67, 'Maths':87}
>>> print(marks.values())
    dict_values([67, 87])
```

PES UNIVERSITY

Use of for and while Loops for Dictionary

1. for loop

Ex: (1)
```
dict = {'a': 'pencil', 'b': ‘eraser', 'c': 'sharpner’}
        for key, value in dict.items():
                print(key, value)
```

Ex: (2)
```
dict = {'a': 'juice', 'b': 'grill', 'c': 'corn’}
        for key in dict:
                print(key, dict[key])
```

PES UNIVERSITY

Use of for and while Loops for Dictionary

2. while loop

Ex:

```
books={"learning python": "Mark Lutz", "think python": "Allen B. Downey",
                "Fluent Python": "Luciano Ramalho"}
key=list(books) #converts keys into a list
i=0
while i<len(key):
        print(key[i],":",books[key[i]])
        i+=1
```

THANK YOU

Mohan Kumar AV, Sindhu Pai

Team Python - 2022

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Dr.Chetana Srinivas and Prof. Sindu R Pai
Department of CSE(AI&ML) and CSE

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Set

Dr.Chetana Srinivas and Prof. Sindu R Pai
Department of CSE(AI & ML) and CSE

Set

Definition:- A set is an unordered collection of Unique elements with zero or more elements present with the following attributes.

- In Python **sets are written with curly braces.**
- **Constructor set()** is used to create an empty set.

Examples :

- Empty_set=set() # Use the **constructor set()**.
- fruitset = { “apple”, “orange”, “kiwi”, “banana”, “cherry”}
- set1 = {(1, 'a', 'hi'), 2, 'hello', 5.56, 3+5j, True}

Characteristics

- **Mutable**: Modifiable
- **Unordered** – Order of elements in a set need not be same as the order in which we add or delete elements to/from the set.
- **Iterable** – A container object capable of returning its members one at a time. Set is eager and not lazy.
- **Not indexable** - Cannot access items in a set by referring to an index, since sets are unordered in nature.
- Check for **membership** using the in operator is faster in case of a set compared to a list, a tuple or a string.

- **Sets are used to :**

o Remove the duplicate elements, inturn allowing us to find the unique elements.

o comparing two iterables for common elements or diference.

Characteristics

- Elements are **unique** – does not support repeated elements.
- Elements should be **hashable.**
- Hashing is a mechanism to convert the given element into an integer.
- An object is hashable if it has a hash value which never changes during its lifetime (it needs a __hash__() method).

Examples of hashable objects:

- int, float, complex, bool, string, tuple, range, frozenset, bytes, decimal.

Examples of Unhashable objects:

- list, dict, set, bytearray

Examples

- #set1={ [1, 'a', 'hi'], 2, 'hello', {3, 4.5, 'how r u' } }
  #TypeError: unhashable type: 'list'

- #set2={ (1, 'a', 'hi'), 2, 'hello', {1:'hi', 2:'hello', 3:'how r u'} }
  #TypeError: unhashable type: 'dict'

- set3={ (1, 'a', 'hi'), 2, 'hello', 5.56, 3+5j, True }
  print(set3)
  #{True, (1, 'a', 'hi'), 2, 5.56, (3+5j), 'hello'}

- my_dict = {'name': 'John', tuple([1,2,3]):'values'}
  print(my_dict)
  #{'name': 'John', (1, 2, 3): 'values'}

Common methods of Set

There are number of **functions built into Python** that takes **set** as the arguments.

- **len()**
- **sum()**
- **sorted()**
- **max()**
- **min()**

Note: Check reversed()

Specific methods of Set

The dir() function returns all properties and methods of the specified object.

- **>>> dir(set)**
- **['__and__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__iand__', '__init__', '__init_subclass__', '__ior__', '__isub__', '__iter__', '__ixor__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__xor__', 'add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersection', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference', 'symmetric_difference_update', 'union', 'update']**

Methods in detail

len()	Returns the number of elements in a container set.	>>> set1={22,33,11,55,44,120}
		>>> len(set1)
		6

max()	Returns the element from Set with maximum value	>>> set1={42,73,25,75,64,145}
		>>> max(set1)
		25

min()	Returns the element from Set with maximum value	>>> set1={25,43,21,66,45,120}
		>>> min(set1)
		21

sum()	Returns the sum of elements of the Set.	>>> set1={4,10,2,7,25}
		>>> sum(set1)
		48

Common methods of Set

sorted()	Returns the sorted sequence or sorted collection in the form of list.	>>> set1={22,33,11,55,44,120}
		>>> sorted(set1)
		[11, 22, 33, 44, 55, 120]
		
		>>> s4={"pes",0,1}
		>>> sorted(s4)
		Traceback (most recent call last):
		File "<stdin>", line 1, in <module>
		TypeError: '<' not supported between instances of 'str' and 'int'

Common operators applied on set

Concatenation (|=) operator can be used to combine two sets together, results in a set that contains items of the two sets.

```
>>> s1={1,2.34,(34,22),"python"}    >>> s2={10,3.22,"programming"}
>>> s1|=s2
>>> s1
{1, 2.34, 'python', (34, 22), 3.22, 'programming', 10}
```

Repetition operator(*) – Throws an error as set has unique elements.

```
>>> s1={1,2,3,4,5,6}
>>> print(s1*3)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for *: 'set' and 'int'
```

Common operators applied on set

Membership operator(in) – returns True if a sequence with the specified value is present in the string otherwise False.

```
>>> s1={1,2,3,4,5,6}
>>> print(1 in s1)
True
>>> print('a' in s1)
False
```

Membership(not in) operator- Returns **True** if the element is not present in the set, else return **False**.

```
>>> s1={1,2.34,(34,22),"python"}
>>> "PYTHON" not in s1
True
>>> (34,22) not in s1
False
```

Specific functions of set

union()- Return the union of sets as a new set. (i.e. all elements that are in either set.) .

```
>>> s1={1,2,3,4,5,6}
>>> s2={6,5,7,8,9,10}
>>> print(s1.union(s2))
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

or

```
>>> print(s1|s2)
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

intersection()- Return the intersection of sets as a new set. (i.e. all elements that are in both sets.) .

```
>>> s1={1,2,3,4,5,6}
>>> s2={6,5,7,8,9,10}
>>> print(s1.intersection(s2))
{5, 6}
```

or

```
>>> print(s1&s2)
{5, 6}
```

Specific functions of set

difference()- Return the difference of two or more sets as a new set.

```
>>> s1={1,2,3,4,5,6}
>>> s2={6,5,7,8,9,10}
>>> print(s1.difference(s2))
{1, 2, 3, 4}
```

or

```
>>> print(s1-s2)
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

add()- adds an element in a container set.

```
>>> s1={1,2,3,4,5,6}
>>> s1.add(7)
>>> s1
{1, 2, 3, 4, 5, 6, 7}
```

```
>>>s1={1,2,3,4,5,6}
>>> s1.add("pes")
>>> s1
{1, 2, 3, 4, 5, 6, "pes"}
```

Specific functions of set

symmetric_difference()- Return a set with the symmetric differences of two sets.

```
>> s1={1,2,3,4,5,6}
>>> s2={6,5,7,8,9,10}
>>> print(s1.symmetric_difference(s2))
{1, 2, 3, 4, 7, 8, 9, 10}
```

or

```
>>> print(s1^s2)
{1, 2, 3, 4, 7, 8, 9, 10}
```

remove()- Removes the specified item in a set. If the item to remove does not exist, **remove() will raise an error.**

```
>>> s1={1,2,3,4,5,6}
>>> s1.remove(5)
>>> print(s1)
{1, 2, 3, 4, 6}
```

or

```
>>> s1.remove(7)
Displays an Error.
```

Specific functions of set

discard()- Removes the specified item in a set. If the item to remove does not exist, **discard() will NOT raise an error.**

```
>>> s1={1,2,3,4,5,6}
>>> s1.remove(6)
>>> print(s1)
{1, 2, 3, 4, 5}
```

or

```
>>> s1.discard(7)
```

Doesn't Displays an Error.

pop()- Removes an item in a set. Sets are unordered, so when using the pop() method, doesn't know which item that gets removed.

```
>>> s1={1,2,3,4,5,6}
>>> s1.pop()
1
>>> print(s1)
{2, 3, 4, 5, 6}
```

Specific functions of set

update()- updates the current set, by adding items from another set (or any other iterable). If the element is present in both the sets, only one appearance of element present in updated set

```
>>> s1={120,12,23}
>>> s1.update([22,15,67,"pes"])
>>> s1
{67, 12, 15, 22, 23, 120, 'pes’}
```

intersection_update()- Removes the items that is not present in both sets .

```
>>> set1={"apple", "banana", "cherry"}
>>> set2={"google", "microsoft", "apple"}
>>> set1.intersection_update(set2)
>>> set1
{'apple'}
```

Specific functions of set

difference_update()- Removes the items that exists in both sets.

```
>>> set1={"apple", "banana", "cherry"}
>>> set2={"google", "microsoft", "apple"}
>>> set1.difference_update(set2)
>>> set1
{'cherry', 'banana’}
```

Symmetric_difference_update()- Updates the original set by removing items that are present in both sets, and inserting the other items.

```
>>> set1={"apple", "banana", "cherry"}
>>> set2={"google", "microsoft", "apple"}
>>> set1.symmetric_intersection_update(set2)
>>> set1
{'microsoft', 'apple', 'cherry', 'google', 'banana'}
```

Specific functions of set

issubset()- Returns True if all items in set ‘x’ are present in set ‘y’.

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

issuperset()- Returns True if all items in set ‘y’ are present in set ‘x’.

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

Specific functions of set

isdisjoint()- Returns True if no items in set s1 is present in set s2.

```
>>> s1={1,2,5}
>>> s2={11,22,55}
>>> s1.isdisjoint(s2)
True
```

clear()- Removes all the elements in a set.

```
>>> x = {"f", "e", "d", "c", "b", "a"}
>>> x.clear()
>>> x
set()
```

Example code 1:

Display the elements of a set one by one in separated by a space between each of them.

```
a = { 10, 30, 10, 40, 20, 50, 30 }

for i in a :

        print(i, end = “  “ )
```

output:40 10 50 20 30

Example code 2:

PES UNIVERSITY

Create a set of numbers from 2 to n.

Steps: create an empty set and add the elements to it.

```
s = set()
n=int(input("enter the value of n")) #5
for i in range(2,n+1):
    s.add(i)
print(s)
```

Output: {14}

Example code 3:

Program to check whether a set is empty or not.

Version 1:

```
s1=set()
if len(s1)==0:
    print("set is empty")
else:
    print("not empty")
```

Version 2:

```
s1=set()
if(s1):
    print("set is empty")
else:
    print(“not empty”)
```

Output: set is empty

Note: Empty data structure is False

THANK YOU

Dr. Chetana Srinivas and Prof.Sindhu R Pai

Team Python - 2022

Department of CSE(AI & ML) and CSE

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Dr. Chetana Srinivas and Prof. Sindu R Pai

Department of CSE(AI&ML) and CSE

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Strings

Dr. Chetana Srinivas and Prof. Sindu R Pai
Department of CSE(AI&ML) and CSE

Strings Processing in python

A **string** is a collection of characters or a sequence of characters .

- Creating strings is as straight forward as assigning a value to a variable.

Ex:

```
var1= ""      # empty string.

var="python"

var="""" this is python""""
```

Characteristics

- A string has zero or more characters.

```
>>> str1=" “                    >>> str1="p"
>>> str2="python“               >>> str3 = “’’python pro “’’
```

- Each character of a string can be accessed by using the index that starts from 0 . Negative indices are allowed.

Example:-

```
>>> str2[0]
      'p’
>>> str2[-1]
      ‘n’
```

- The index value can be an expression that is computed.

```
>>> str2[2+3]
      'n’
```

Characteristics

- **Immutable** – cannot modify the individual characters in a string. One cannot **add, delete, or replace** characters of a string.

Example:-

```
str2="python"
>>> str2[2]='T'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

- **Iterable -**

```
for i in str2:
        print(i)
```

PES UNIVERSITY

All string operations that "modify" a string return a new string that is a modified version of the original string.

Common methods of String

- **The len(str)** returns the number of characters in a string.

```
>>> str1="python"
>>> len(str1)
6
```

- **s.index(chr)** returns the index of the first occurrence of chr in s.

```
>>> str1="python programming“
>>> str1.index(‘p’)
0
```

```
>>> str1.index('prog’)
7
```

- **count() -** Returns the number of times a specified value / character occurs in a string.

```
>>> str1="Welcome to Python Class“
>>> str1.count('s’)
2
```

Common methods of String

- **len(str)** returns the number of characters in a string.

```
>>> str1="python"
>>> len(str1)
6
```

min and max functions as applied to strings return the smallest and largest character respectively based on the underlying Unicode encoding.

For example, all lowercase letters are larger have a larger Unicode value than all uppercase letters.

- **Example:-**

```
>>> str1="Python“
>>> min(str1)
'P’
>>> max(str1)
'y'
```

Common operators applied on string

Concatenation (+) operator can be used to add multiple strings together.

```
>>> str1="Python“          >>> str2="Language"
>>> str1+str2
'PythonLanguage’
```

Repetition (*) operator returns is used to repeat the string to a certain length.

```
>>> str1="Python Programming"
>>> str1*2
'Python ProgrammingPython Programming'
```

Common operators applied on string

Scope resolution (::) operator assigns the characters of string str1 into an another string str2.

```
>>> str1="Python Programming"
>>> str2=str1[::]      // same as str2=str1
>>> str2
'Python Programming’
>>> id(str1)
1534440206336
>>> id(str2)
1534440206336
```

Common operators applied on string

Membership (in) operator returns True if a sequence with the specified value is present in the string otherwise False.

```
>>> str2="Language"
>>> "Lang" in "Language"
True
>>> "Png" in "Language"
False
```

Membership (not in) operator returns True if a sequence with the specified value is is not present in the string otherwise False.

.

```
>>> str1="Language"
>>> "LAN" not in str1
True
>>> "Lang" not in str1
False
```

Common operators applied on string

The slice operator s[start:end] returns the substring starting with index start, up to but not including index end.

- **Example:-**

```
>>> s = 'Monty Python’
>>> s[0:4]
    Mont
>>> s[6:7]
    P
>>> s[6:20]
    Python
```

Specific methods of String

- **The dir() function** returns all properties and methods of the specified object.

```
>>>dir(str)
```

['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

Specific methods of String

- **index(chr)** returns the index of the first occurrence of chr in s.

```
>>> str1="python programming“
>>> str1.index('prog’)
>>> str1.index(‘p’)
0
```

- **count() -** Returns the number of times a specified value / character occurs in a string.

```
>>> str1="Welcome to Python Class“
>>> str1.count('s’)
2
```

Specific functions of string

startswith(prefix,start,end) returns True if string starts with the given prefix otherwise returns False.

prefix – A string that needs to be checked.

start – starting position where prefix is needed to be checked within the string.

end- Ending position where prefix is needed to be checked within the string.

Ex:-

```
>>> str1.startswith("W",0)
True
>>> str1.startswith("W",1)
False
>>> str1.startswith("App",5,8)
True
>>> str1.startswith("App",6,9)
False
```

Specific functions of string

endswith(search_string,start,end) returns True if original string ends with the search_string otherwise returns False.

search_string – A string to be searched.

start – starting position of the string from where the search_string is to be searched.

end- Ending position of the string to be considered for searching.

Ex:-

```
>>> str1="Welcome to Python Class“
>>> str1.endswith("Class")
>>> str1.endswith("Class")
True
>>> str1.endswith("Class",18)
True
>>> str1.endswith("Class",19)
False
>>> str1.endswith("Class",18,22)
False
>>> str1.endswith("Class",18,23)
True
```

Specific functions of string

- There are a number of methods specific to strings in addition to the general sequence operations.
- **String processing involves search**.
- The **find(substring,start,end) method** returns the index location of the first occurrence of a specified substring. If not found, returms -1.

  Substring- The substring to search for.

  start – where to start the search. Default is 0.

  end – where to end the search. Default is 0.

```
>>> s3="chocolate“
>>> s3.find("c")
0
>>> s3.find("z")
-1
```

```
>>>s3.find(“c”,4,9)
-1

>>>s3.find(“c”,3,9)
3
```

Specific functions of string

- The **rfind(substring,start,end) method** returns the index location of the **last occurrence** of a specified substring. If not found, returns -1.

  Substring- The substring to search for.

  start – where to start the search. Default is 0.

  end – where to end the search. Default is 0.

```
>>> s3="chocolate“
>>> s3.rfind("c")
3
>>> s3.rfind("c",-6,-1)
3
>>> s3.rfind("c",4,9)
-1
>>>str1.rfind("c",0)
3
```

```
>>> s3.rfind("c",1,2)
-1
```

Specific functions of string

- The **strip(characters) method** removes characters from both left and right based on the argument.
- The l**strip(characters) method** removes characters from left based on the argument.
- The r**strip(characters) method** removes characters from right based on the argument.

```
>>> str1="          Welcome to Python class          "
>>> str1.strip()
'Welcome to Python class’
 >>> str1.lstrip()
 'Welcome to Python class          ‘
 >>> str1.rstrip()
 '          Welcome to Python class'
```

Specific functions of string

- >>> str1="madam"
  >>> str1.strip('m')
  'ada'
- >>> str1.lstrip('m')
  'adam'
- >>> str1.rstrip('m')
  'mada'
- >>> str1.rstrip('ma')
  'mad'
- >>> str1.rstrip('maz')
  'mad'
- >>> str1.rstrip('mjaz')
  'mad'

Specific functions of string

- The **replace method** produces a new string with every occurrence of a given substring within the original string replaced with another.
- >>> s1.replace("a","b")
  'whbtsbpp'
- >>> s1.replace("a","b",1)
  'whbtsapp'
- >>> s1.replace("a","b",2)
  'whbtsbpp'
- >>> s1.replace("a","b",0)
  'whatsapp'
- >>> s1.replace("at","b",1)
  'whbsapp'
- >>> s1.replace("at","b",2)
  'whbsapp'

Specific functions of string

- The **title() method** returns a string where the first character in every word is upper case.
- >>> str1="people education society"
  >>> str1.title()
  'People Education Society'

- The **capitalize() method** returns a string where the first character is upper case, and the rest is lower case.
  >>> s1="Python"
  >>> s1.capitalize()
  'Python'

Specific functions of string

The **join() method** takes all items in an iterable and joins them into one string.

- A string must be specified as the separator.

Example:-

```
>>> s2="abc"
>>> s1="abc"
>>> s2="xyz"
>>> s1.join(s2)
'xabcyabcz’
```

Specific functions of string

The **split()** method splits a string into a list.

Example:-

- >>> str1="When someone is lost, dare to help them find the way."
  >>> str1.split()
  ['When', 'someone', 'is', 'lost,', 'dare', 'to', 'help', 'them', 'find', 'the', 'way.']
- >>> str1.split("i")
  ['When someone ', 's lost, dare to help them f', 'nd the way.']
- >>> str1.split("i",1)
  ['When someone ', 's lost, dare to help them find the way.']
- >>> str1.split("i",1,2) Traceback (most recent call last): File "<stdin>", line 1, in <module> TypeError: split() takes at most 2 arguments (3 given)

Specific functions of string

- The **isspace()** method returns “True” if all characters in the string are whitespace characters, Otherwise, It returns “False”.

```
>>> s1=“  “
>>> s1.isspace()
True
```

```
>>> s1="543a"
>>> s1.isspace()
False
```

- The **ljust()** method will left align the string, using a specified character and returns a new string

```
>>> s3="chocolate"
>>> s3.ljust(20)
'chocolate          ‘
>>> s3.ljust(20,"#")
'chocolate##########'
```

Specific functions of string

- The **rjust()** method will right align the string, using a specified character and returns a new string

```
>>> s3="chocolate"
>>> s3.rjust(20)
'          chocolate’
>>> s3.rjust(20,"$")
'$$$$$$$$$$chocolate'
```

- The **center()** method will center align the string, using a specified character and returns a new string.

```
>>> s3="chocolate"
>>> s3.center(20)
'    chocolate     ‘
>>> s3.center(20,'*’)
'*****chocolate******'
```

Specific functions of string

- The **zfill()** method adds zeros (0) at the beginning of the string, until it reaches the specified length.

```
>>> s3.zfill(10)
'0chocolate’
>>> s3.zfill(20)
'00000000000chocolate’
```

- The **isnumeric()** method returns True if all the characters are numeric (0-9), else False.

```
>>> s1="543“
>>> s1.isnumeric()
True
```

```
>>> s1="543a"
>>> s1.isnumeric()
False
```

Specific functions of string

Checking the Contents of a String			
`str.isalpha()`	Returns True if *str* contains only letters.	`s = 'Hello'`	`s.isalpha()` → `True`
		`s = 'Hello!'`	`s.isalpha()` → `False`
`str.isdigit()`	Returns True if *str* contains only digits.	`s = '124'`	`s.isdigit()` → `True`
		`s = '124A'`	`s.isdigit()` → `False`
`str.islower()` `str.isupper()`	Returns True if *str* contains only lower (upper) case letters.	`s = 'hello'`	`s.islower()` → `True`
		`s = 'Hello'`	`s.isupper()` → `False`
`str.lower()` `str.upper()`	Return lower (upper) case version of *str*.	`s = 'Hello!'`	`s.lower()` → `'hello!'`
		`s = 'hello!'`	`s.upper()` → `'HELLO!'`
Searching the Contents of a String			
`str.find(w)`	Returns the index of the first occurrence of `w` in *str*. Returns -1 if not found.	`s = 'Hello!'`	`s.find('l')` → `2`
		`s = 'Goodbye'`	`s.find('l')` → `-1`
Replacing the Contents of a String			
`str.replace(w,t)`	Returns a copy of *str* with all occurrences of `w` replaced with t.	`s = 'Hello!'`	`s.replace('H', 'J')` → `'Jello'`
		`s = 'Hello'`	`s.replace('ll', 'r')` → `'Hero'`
Removing the Contents of a String			
`str.strip(w)`	Returns a copy of *str* with all leading and trailing characters that appear in w removed.	`s = ' Hello! '` `s = 'Hello\n'`	`s.strip(' !')` → `'Hello'` `s.strip('\n')` → `'Hello'`
Splitting a String			
`str.split(w)`	Returns a list containing all strings in *str* delimited by w.	`s = 'Lu, Chao'`	`s.split(',')` → `['Lu', 'Chao']`

Try these functions!!

- encode()
- decode()
- maketrans()
- translate()
- partition()

THANK YOU

Dr. Chetana Srinivas, Prof.Sindhu R Pai

Team Python - 2022

Department of CSE(AI & ML) and CSE

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Mohan Kumar AV, Sindhu Pai
Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Files

Mohan Kumar AV, Sindhu Pai
Department of Computer Science and Engineering

Different ways of giving input to the program:

- Using command line arguments.
- Through the keyboard , using input() function.

In both the cases the amount of input given would be minimal

and also prone to errors.

This is where the files comes in to picture.

Files

Advantages of Files:

- Data is persistent even after the termination of the program.
- The data set can be much larger.
- The data can be input much more quickly and with less chance of error.

Python, a file operation takes place in the following order:

1.Open a file

2.Read or write (perform operation)

3.Close the file

Files

Functions:

open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

file is either a text or byte string giving the name (and the path if the file isn't in the current working directory) of the file to be opened.

mode is an optional string that specifies the mode in which the file is opened. It defaults to 'r' which means open for reading in text mode. Other modes are ‘a’, ‘a+’, etc.

Lets not worry about other parameters.

Files

PES UNIVERSITY

Opening a file:

- A file opening may not always succeed. If it fails, it throws an exception.
- A few possible exceptions:
  - opening a non-existent file for reading.
  - opening a file for writing where directory does not allow us to create files.

- If the required file exists then, a file object is returned.

- OS provides the required resources when the file gets created.

PES UNIVERSITY

read()

After we open a file, we use the read() method to read its contents. File can be read in different ways:

- read() - returns the contents of the file.
- readline() - reads one line from the file and returns it as a string. The string returned by readline will contain the newline character at the end.
- readlines() - returns a list containing each line in the file as a list item.

PES UNIVERSITY

Writing to a file:

- In order to write it to a file we may use:

  - write()

  - print()

- close() - We return the resources utilized back to OS by calling a function called close on the open file.

Files

Example:

```
file1 = open("test.txt", "r") # open a file

read_content = file1.read() # read the file

print(read_content)

file1.close() # close the file
```

THANK YOU

Mohan Kumar AV, Sindhu Pai

Team Python - 2022

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Dr. Chetana Srinivas and Prof. Sindhu R Pai,
Department of CSE (AI&ML) and CSE

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions – Position and Keyword Arguments

Dr. Chetana Srinivas and Prof. Sindhu R Pai,

Department of CSE(AI & ML) and CSE

Function Call: Keyword and Positional Arguments

PES UNIVERSITY

When we call a function in Python, we can pass in two different types of arguments

1. **Keyword arguments (named arguments) –** Are arguments passed to a function or method which is preceded by a keyword (parameter_name)and an equals sign. Argument name must be same as the parameter name in the function definition.

2. **Positional arguments –** Are arguments that need to be included in the proper position or order.

Note: If there are both keyword and positional arguments, Keyword arguments must follow positional arguments

Function Call: Keyword and Positional Arguments

PES UNIVERSITY

Differences

Keyword arguments	Positional arguments
Parameter names are used to pass the argument during the call	Arguments are passed in the order of parameters. The order defined in the order function declaration.
Order of parameter Names can be changed to pass the argument(or values).	Order of values cannot be changed to avoid the unexpected output.
Syntax : – FunctionName(paramName=value,…)	Syntax : – FunctionName(value1,value2,value3…)

Function Call: Keyword and Positional Arguments

Keyword arguments (named arguments)

Example 1: -

```
def nameAge(name, age):
    print("Hi, I am",name)
    print("My age is ",age)
nameAge(name="Rajeev", age=26)
nameAge(age=26, name="Rajeev")
```

Positional arguments

Example 2: -

```
def minus(a,b):
    return a-b

X=20;Y=10
print("Difference between two numbers is=",minus(X,Y))
```

Combination of both Positional and keyword arguments

Rule: All keyword arguments must follow positional arguments

Example 3:-

```
def f1(x, y, z, a, b):
    print(a, b, x, y, z)
f1(3,5,z = {4,5},b = [12,7,8,4], a = 99.7) # 99.7 [12, 7, 8, 4] 3 5 {4, 5}
#f1(3,5,x = {4,5},b = [12,7,8,4], a = 99.7) #TypeError: f1() got multiple values for argument 'x'
#f1(x = 9, y = 99,2,3,b = 22) #SyntaxError: positional argument follows keyword argument
```

Function Call: Default Parameters

Default parameters are always a part of the symbol table which is added during the leader processing phase. If the user did not send the argument, then this default parameter is used in the processing.

Example 4:-

```
def f1(a,b=5):
    print(a,b)
f1(4)
f1(4,13)
```

```
C:\Users\Dell>python notes_functions.py
4 5
4 13

C:\Users\Dell>
```

Example 5:-

```
x=12
def f1(a,b=x):
    print(a,b)
f1(4)
f1(4,13)
```

```
C:\Users\Dell>python notes_functions.py
4 12
4 13

C:\Users\Dell>
```

THANK YOU

Dr. Chetana Srinivas and Prof.Sindhu R Pai

Team Python - 2022

Department of CSE(AI & ML) and CSE

Email-id :- chetanasrinivas@pes.edu

sindhurpai@pes.edu

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Dr. Chetana Srinivas and Prof. Sindhu R Pai,
Department of CSE (AI&ML) and CSE

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions – Variable number of Arguments

Dr. Chetana Srinivas and Prof. Sindhu R Pai,

Department of CSE(AI & ML) and CSE

Function Call: Arguments and Parameters

- In the function definition, variables are specified which receive the arguments when the function is called or invoked. **These are called parameters**.
- The parameters are **always variables**.
- When the function call is made, the control is transferred to the function definition. The arguments are evaluated and copied to the corresponding parameters.
- This is called **parameter passing by value or call by value**.
- The value of arguments are passed by value to the parameters.

Function Call: Arguments and Parameters

- When a function performs a specified task, it may require some values to operate on.
- These values have to be provided when calling a function as input.
- These values have to be put within the pair of round parentheses in the function call.
- **They are called as arguments**.

Function: Global and Local variables

PES UNIVERSITY

There are two types of variables

- **Global Variables:** Variables created outside all functions. Scope of Global variable is within the file and outside that file too.
- **Local variables:** Variables created inside the function and accessible only to that function. Scope of the local variable is within the function.

Example 1:

```
x=10
print("global first outside",x)
def f1():
    x=11
    #local variable x.
   #Life and scope is only within this function
   print("inside",x)
f1()
print("global second outside",x)
```

```
C:\Users\Dell>python notes_functions.py
global first outside 10
inside 11
global second outside 10

C:\Users\Dell>
```

Function: Global and Local variables

Example 2: Global variable is read-only inside a function. We cannot modify the value of that variable inside the function directly.

```
x=10
print(x)
def f1():
    x=x+1 #UnboundLocalError.
    print("inside",x)
f1()
print("outside",x)
```

Function: Global and Local variables

Example 3: Usage of global keyword

```
x=10
print("global first outside",x)
def f1():
    global x
    x=x+1
    print("inside",x)
f1()
print("global second outside",x)
#If you interchange the first two statements inside the function, it results in Error.
```

```
C:\Users\Dell>python notes_functions.py
global first outside 10
inside 11
global second outside 11
```

Example 4:- Calculate the sum of two numbers given and return the sum.

```
def add_numbers(x,y):
    sum = x + y
    return sum

output = add_Numbers(10,5)
print("The sum of two numbers is = ",output)
```

Output:

The sum of two numbers is = 15

Function Call: Arguments and Parameters

Example 5:- Calculate the area of triangle given the dimensions.

```
def area_tri(b,h) :
    a=0.5*b*h
    return a
Area=area_tri(5,6)
print("area=",Area)
```

- The function definition has two parameters, which indicate that it requires two arguments to be passed when the function is called/invoked.
- The arguments are constants in this example, 5 and 6.
- These values are copied to the parameters b and h respectively.

Function Call: Arguments and Parameters

When a function is called, it is imperative that the number of arguments passed **MUST ALWAYS MATCH** the number of parameters in the function definition

Example 6:-

```
def  product(x,y,z) :
        a=x*y*z return a
r=product(5,6,2)
#r=product(5) #error : too few arguments
#r=product(5,6,2,7) #error : too many arguments
print(“product =",r)
```

Output:

product = 60

Function Call: Categories of Functions

1. No arguments : No return value

```
def add():
        a = 10
        b = 20
        print(a+b)
add()
```

Output: 30

1. No arguments : with return value

```
def add()
    a=10
    b=20
    return a+b
sum = add()
print(sum)
```

Output: 30

Function Call: Categories of Functions

3. With arguments : No return value

```
def add(a,b):
        print(a+b)
add(10,20)
```

Output:
30

4. With arguments : With return value

```
def add(a,b):
    return a+b
sum = add(10,20)
print(sum)
```

Output:
30

Variable number of positional arguments: *arg [Any variable can be used with *]

- * handles the variable number of positional arguments in the function.
- arg is of type tuple inside the function.

Example 7:

```
def f1(*arg):
    print(arg, type(arg)) #arg is of type tuple
    for i in arg:
        print(i)
f1(2,1,6,4,9,7) #All three function calls are valid
f1()
f1(2,4)
```

```
C:\Users\Dell>python notes_functions.py
(2, 1, 6, 4, 9, 7) <class 'tuple'>
2
1
6
4
9
7
() <class 'tuple'>
(2, 4) <class 'tuple'>
2
4

C:\Users\Dell>
```

Variable number of keyword arguments: **kwarg [Any variable can be used with **]

- ** handles the variable number of keyword arguments in the function.
- kwarg is of type dictionary inside the function.

Example 8:

```
def f1(**kwarg):
    print(kwarg, type(kwarg)) #kwarg is of type dict
    for i in kwarg:
        print(i,end="")
    print()
    for i in kwarg.keys():
        print(i,end="")
    print()
    for i in kwarg.values():
        print(i,end="")
f1(a=2,b=1,c=6)
```

```
C:\Users\Dell>python notes_functions.py
{'a': 2, 'b': 1, 'c': 6} <class 'dict'>
a b c
a b c
2 1 6
C:\Users\Dell>
```

THANK YOU

Dr. Chetana Srinivas and Prof.Sindhu R Pai

Team Python - 2022

Department of CSE(AI & ML) and CSE

Email-id :- chetanasrinivas@pes.edu

sindhurpai@pes.edu