* List and its default function.

List is the most versatile data-type available in Python that can be written as a collection of comma-separated values or items between square brackets. The items in a list need not necessarily be homogeneous, i.e. of the same type. This property of List makes it one of the most powerful tools in Python eg:[‘HELLO’, 57, “SKY”]. A single list can contain different Data-Types such as integers, strings, as well as Objects. Lists are mutable, and hence can be changed even after their creation.

1. Creating a list:

list1 = ['jack', 'jill', 1998, 2019];

list2 = [1, 2,3, 4, 5 ];

list3 = ["w", "x", "y", "z"]

1. Adding value to list using append():

Using the built-in append() function elements can be added to the List. At a time only one element can be added to the list by the use of append() method.

a = ["bee", "moth"]

print(a)

a.append("ant")

print(a)

RESULT:

['bee', 'moth']

['bee', 'moth', 'ant']

1. Adding value to list using insert():

Using the insert() method, elements can be added to the list at your desired position.

a = ["bee", "moth"]

print(a)

A.insert(0,"ant")

print(a)

RESULT:

['bee', 'moth']

['ant','bee', 'moth']

1. Removing value from the list:

using the built-in remove() function, elements can be removed from the List but an Error will arise if the element is not present in the set. Remove() method is only capable of removing one element at a time.

a = ["bee", "moth", "ant"]

print(a)

a.remove("moth")

print(a)

RESULT

['bee', 'moth', 'ant']

['bee', 'ant']

1. pop():

Removes the item at the given position in the list, and returns it. If no index is specified, pop() removes and returns the last item in the list.

# Example 1: No index specified

a = ["bee", "moth", "ant"]

print(a)

a.pop()

print(a)

# Example 2: Index specified

a = ["bee", "moth", "ant"]

print(a)

a.pop(1)

print(a)

RESULT

['bee', 'moth', 'ant']

['bee', 'moth']

['bee', 'moth', 'ant']

['bee', 'ant']

1. reverse():

Reverses the elements of the list in place.

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

a.reverse()

print(a)

a = ["bee", "wasp", "moth", "ant"]

a.reverse()

print(a)

RESULT

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

['ant', 'moth', 'wasp', 'bee']

* Dictionary and its default function.

Creating a dictionary is as simple as placing items inside curly braces {} separated by commas.

# empty dictionary

my\_dict = {}

# dictionary with integer keys

my\_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys

my\_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()

my\_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair

my\_dict = dict([(1,'apple'), (2,'ball')])

* Accessing Elements from Dictionary

# get vs [] for retrieving elements

my\_dict = {'name': 'Jack', 'age': 26}

# Output: Jackprint(my\_dict['name'])

# Output: 26print(my\_dict.get('age'))

# Trying to access keys which doesn't exist throws error# Output Noneprint(my\_dict.get('address'))

# KeyErrorprint(my\_dict['address'])

Output

Jack

26

None

Traceback (most recent call last):

File "<string>", line 15, in <module>

print(my\_dict['address'])

KeyError: 'address'

* Changing and Adding Dictionary elements

# Changing and adding Dictionary Elements

my\_dict = {'name': 'Jack', 'age': 26}

# update value

my\_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}print(my\_dict)

# add item

my\_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}print(my\_dict)

Output

{'name': 'Jack', 'age': 27}

{'name': 'Jack', 'age': 27, 'address': 'Downtown'}

* Removing elements from Dictionary

# Removing elements from a dictionary

# create a dictionary

squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# remove a particular item, returns its value# Output: 16print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}print(squares)

# remove an arbitrary item, return (key,value)# Output: (5, 25)print(squares.popitem())

# Output: {1: 1, 2: 4, 3: 9}print(squares)

# remove all items

squares.clear()

# Output: {}print(squares)

# delete the dictionary itselfdel squares

# Throws Errorprint(squares)

Output

16

{1: 1, 2: 4, 3: 9, 5: 25}

(5, 25)

{1: 1, 2: 4, 3: 9}

{}

Traceback (most recent call last):

File "<string>", line 30, in <module>

print(squares)

NameError: name 'squares' is not defined

* Sets and its default function.

A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.

* Creating Python Sets

# Different types of sets in Python# set of integers

my\_set = {1, 2, 3}print(my\_set)

# set of mixed datatypes

my\_set = {1.0, "Hello", (1, 2, 3)}print(my\_set)

Output

{1, 2, 3}

{1.0, (1, 2, 3), 'Hello'}

* Creating an empty set

Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements, we use the set() function without any argument.

# Distinguish set and dictionary while creating empty set

# initialize a with {}

a = {}

# check data type of aprint(type(a))

# initialize a with set()

a = set()

# check data type of aprint(type(a))

Output

<class 'dict'>

<class 'set'>

Modifying a set in Python

We can add a single element using the add() method, and multiple elements using the update() method. The update() method can take [tuples](https://www.programiz.com/python-programming/tuple), lists, [strings](https://www.programiz.com/python-programming/string) or other sets as its argument. In all cases, duplicates are avoided.

# initialize my\_set

my\_set = {1, 3}print(my\_set)

# if you uncomment line 9,# you will get an error# TypeError: 'set' object does not support indexing

# my\_set[0]

# add an element# Output: {1, 2, 3}

my\_set.add(2)print(my\_set)

# add multiple elements# Output: {1, 2, 3, 4}

my\_set.update([2, 3, 4])print(my\_set)

# add list and set# Output: {1, 2, 3, 4, 5, 6, 8}

my\_set.update([4, 5], {1, 6, 8})print(my\_set)

Output

{1, 3}

{1, 2, 3}

{1, 2, 3, 4}

{1, 2, 3, 4, 5, 6, 8}

* Removing elements from a set

# Difference between discard() and remove()

# initialize my\_set

my\_set = {1, 3, 4, 5, 6}print(my\_set)

# discard an element# Output: {1, 3, 5, 6}

my\_set.discard(4)print(my\_set)

# remove an element# Output: {1, 3, 5}

my\_set.remove(6)print(my\_set)

# discard an element# not present in my\_set# Output: {1, 3, 5}

my\_set.discard(2)print(my\_set)

# remove an element# not present in my\_set# you will get an error.# Output: KeyError

my\_set.remove(2)

Output

{1, 3, 4, 5, 6}

{1, 3, 5, 6}

{1, 3, 5}

{1, 3, 5}

Traceback (most recent call last):

File "<string>", line 28, in <module>

KeyError: 2

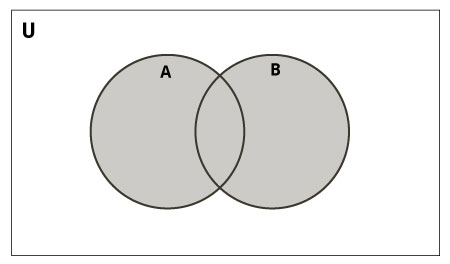
* Python Set Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.

Let us consider the following two sets for the following operations.

>>> A = {1, 2, 3, 4, 5}>>> B = {4, 5, 6, 7, 8}

* Set Union



Set Union in Python

Union of A and B is a set of all elements from both sets.

Union is performed using | operator. Same can be accomplished using the union() method.

# Set union method# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use | operator# Output: {1, 2, 3, 4, 5, 6, 7, 8}print(A | B)

Output

{1, 2, 3, 4, 5, 6, 7, 8}

Try the following examples on Python shell.

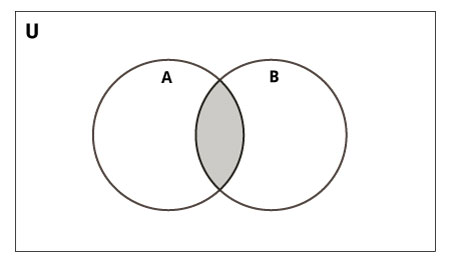
# use union function>>> A.union(B)

{1, 2, 3, 4, 5, 6, 7, 8}

# use union function on B>>> B.union(A)

{1, 2, 3, 4, 5, 6, 7, 8}

* Set Intersection



Set Intersection in Python

Intersection of A and B is a set of elements that are common in both the sets.

Intersection is performed using & operator. Same can be accomplished using the intersection() method.

# Intersection of sets# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use & operator# Output: {4, 5}print(A & B)

Output

{4, 5}

Try the following examples on Python shell.

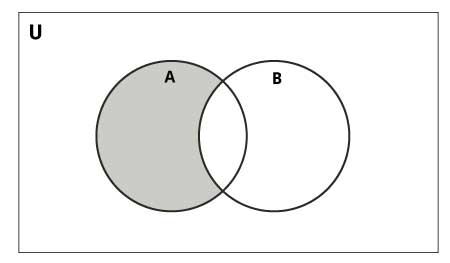
# use intersection function on A>>> A.intersection(B)

{4, 5}

# use intersection function on B>>> B.intersection(A)

{4, 5}

* Set Difference



Set Difference in Python

Difference of the set B from set A(A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of elements in B but not in A.

Difference is performed using - operator. Same can be accomplished using the difference() method.

# Difference of two sets# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use - operator on A# Output: {1, 2, 3}print(A - B)

Output

{1, 2, 3}

Try the following examples on Python shell.

# use difference function on A>>> A.difference(B)

{1, 2, 3}

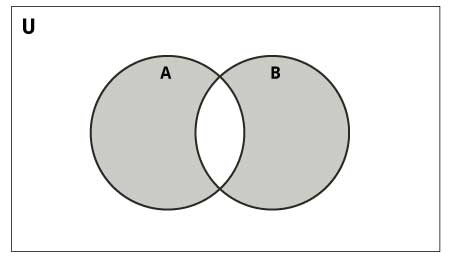
# use - operator on B>>> B - A

{8, 6, 7}

# use difference function on B>>> B.difference(A)

{8, 6, 7}

* Set Symmetric Difference



Set Symmetric Difference in Python

Symmetric Difference of A and B is a set of elements in A and B but not in both (excluding the intersection).

Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric\_difference().

# Symmetric difference of two sets# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use ^ operator# Output: {1, 2, 3, 6, 7, 8}print(A ^ B)

Output

{1, 2, 3, 6, 7, 8}

Try the following examples on Python shell.

# use symmetric\_difference function on A>>> A.symmetric\_difference(B)

{1, 2, 3, 6, 7, 8}

# use symmetric\_difference function on B>>> B.symmetric\_difference(A)

{1, 2, 3, 6, 7, 8}

* Tuple and explore default methods.
* Creating a Tuple

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them.

# Different types of tuples

# Empty tuple

my\_tuple = ()print(my\_tuple)

# Tuple having integers

my\_tuple = (1, 2, 3)print(my\_tuple)

# tuple with mixed datatypes

my\_tuple = (1, "Hello", 3.4)print(my\_tuple)

# nested tuple

my\_tuple = ("mouse", [8, 4, 6], (1, 2, 3))print(my\_tuple)

Output

()

(1, 2, 3)

(1, 'Hello', 3.4)

('mouse', [8, 4, 6], (1, 2, 3))

A tuple can also be created without using parentheses. This is known as tuple packing.

my\_tuple = 3, 4.6, "dog"print(my\_tuple)

# tuple unpacking is also possible

a, b, c = my\_tuple

print(a) # 3print(b) # 4.6print(c) # dog

Output

(3, 4.6, 'dog')

3

4.6

Dog

* Creating a tuple with one element

Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is, in fact, a tuple.

my\_tuple = ("hello")print(type(my\_tuple)) # <class 'str'>

# Creating a tuple having one element

my\_tuple = ("hello",)print(type(my\_tuple)) # <class 'tuple'>

# Parentheses is optional

my\_tuple = "hello",print(type(my\_tuple)) # <class 'tuple'>

Output

<class 'str'>

<class 'tuple'>

<class 'tuple'>

* Access Tuple Elements

There are various ways in which we can access the elements of a tuple.

1. Indexing

We can use the index operator [] to access an item in a tuple, where the index starts from 0.

So, a tuple having 6 elements will have indices from 0 to 5. Trying to access an index outside of the tuple index range(6,7,... in this example) will raise an IndexError.

The index must be an integer, so we cannot use float or other types. This will result in TypeError.

Likewise, nested tuples are accessed using nested indexing, as shown in the example below.

# Accessing tuple elements using indexing

my\_tuple = ('p','e','r','m','i','t')

print(my\_tuple[0]) # 'p' print(my\_tuple[5]) # 't'

# IndexError: list index out of range# print(my\_tuple[6])

# Index must be an integer# TypeError: list indices must be integers, not float# my\_tuple[2.0]

# nested tuple

n\_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested indexprint(n\_tuple[0][3]) # 's'print(n\_tuple[1][1]) # 4

Output

p

t

s

4

2. Negative Indexing

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and so on.

# Negative indexing for accessing tuple elements

my\_tuple = ('p', 'e', 'r', 'm', 'i', 't')

# Output: 't'print(my\_tuple[-1])

# Output: 'p'print(my\_tuple[-6])

Output

t

p

3. Slicing

We can access a range of items in a tuple by using the slicing operator colon :.

# Accessing tuple elements using slicing

my\_tuple = ('p','r','o','g','r','a','m','i','z')

# elements 2nd to 4th# Output: ('r', 'o', 'g')print(my\_tuple[1:4])

# elements beginning to 2nd# Output: ('p', 'r')print(my\_tuple[:-7])

# elements 8th to end# Output: ('i', 'z')print(my\_tuple[7:])

# elements beginning to end# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')print(my\_tuple[:])

Output

('r', 'o', 'g')

('p', 'r')

('i', 'z')

('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need the index that will slice the portion from the tuple.



Element Slicing in Python

* Changing a Tuple

Unlike lists, tuples are immutable.

This means that elements of a tuple cannot be changed once they have been assigned. But, if the element is itself a mutable data type like list, its nested items can be changed.

We can also assign a tuple to different values (reassignment).

# Changing tuple values

my\_tuple = (4, 2, 3, [6, 5])

# TypeError: 'tuple' object does not support item assignment# my\_tuple[1] = 9

# However, item of mutable element can be changed

my\_tuple[3][0] = 9 # Output: (4, 2, 3, [9, 5])print(my\_tuple)

# Tuples can be reassigned

my\_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')print(my\_tuple)

Output

(4, 2, 3, [9, 5])

('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

We can use + operator to combine two tuples. This is called concatenation.

We can also repeat the elements in a tuple for a given number of times using the \* operator.

Both + and \* operations result in a new tuple.

# Concatenation# Output: (1, 2, 3, 4, 5, 6)print((1, 2, 3) + (4, 5, 6))

# Repeat# Output: ('Repeat', 'Repeat', 'Repeat')print(("Repeat",) \* 3)

Output

(1, 2, 3, 4, 5, 6)

('Repeat', 'Repeat', 'Repeat')

* Deleting a Tuple

As discussed above, we cannot change the elements in a tuple. It means that we cannot delete or remove items from a tuple.

Deleting a tuple entirely, however, is possible using the keyword [del](https://www.programiz.com/python-programming/keyword-list" \l "del).

# Deleting tuples

my\_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# can't delete items# TypeError: 'tuple' object doesn't support item deletion# del my\_tuple[3]

# Can delete an entire tupledel my\_tuple

# NameError: name 'my\_tuple' is not definedprint(my\_tuple)

Output

Traceback (most recent call last):

File "<string>", line 12, in <module>

NameError: name 'my\_tuple' is not defined

* Tuple Methods

Methods that add items or remove items are not available with tuple. Only the following two methods are available.

Some examples of Python tuple methods:

my\_tuple = ('a', 'p', 'p', 'l', 'e',)

print(my\_tuple.count('p')) # Output: 2print(my\_tuple.index('l')) # Output: 3

Output

2

3

* String and explore default methods.

A string is a sequence of characters.

A character is simply a symbol. For example, the English language has 26 characters.

Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0s and 1s.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encodings used.

In Python, a string is a sequence of Unicode characters. Unicode was introduced to include every character in all languages and bring uniformity in encoding.

# defining strings in Python# all of the following are equivalent

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)

When you run the program, the output will be:

Hello

Hello

Hello

Hello, welcome to

the world of Python