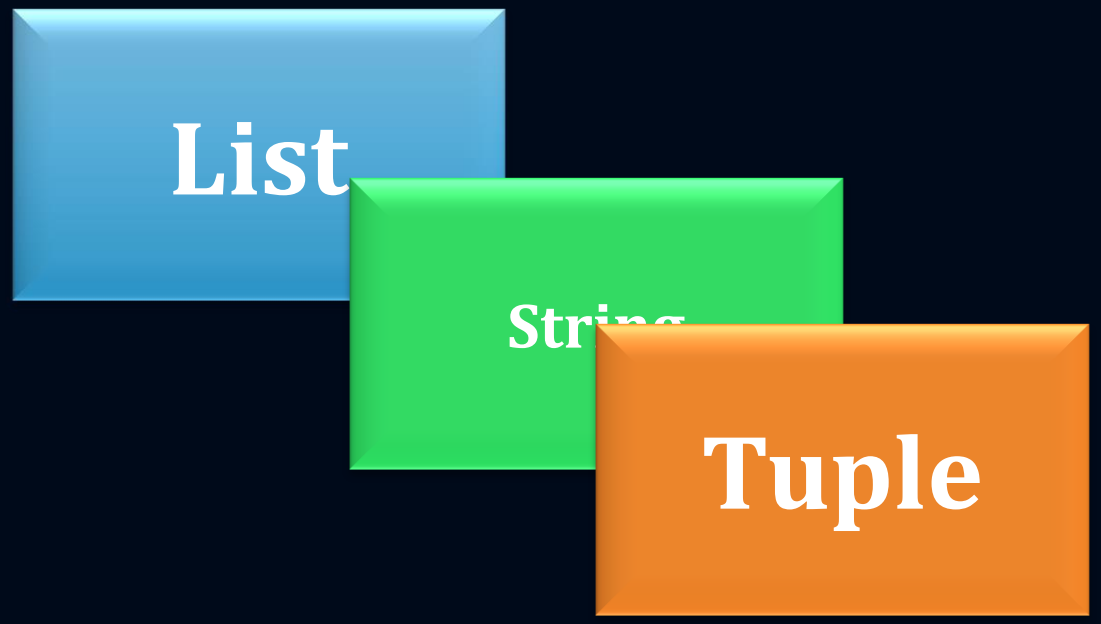
Python Programming

Day 4: Sequence Data Types

Sequence Data Types



Color and symbol meaning

Hint

Preferred

Student's activity

	Python Keyword
	In-built functions
	Strings
	Output

List

A list is an ordered set of values, where each value is identified by an index.

[10, 20, 30, 40]

["spam", "bungee", "swallow"]

Lists are similar to strings, which are ordered sets of characters, except that the elements of a list can have any type.

List

There are several ways to create a new list

X = list ( )

X = [‘a’, 25, ‘dog’, 3.142]

List

The following list contains a string, a float, an integer, and another list:

["hello", 2.0, 5, [10, 20]] ← **Nested list**

Finally, there is a special list that contains no elements. It is called the **empty list**, and is denoted [ ].

List

List assignment examples

```
>>> vocabulary = [“ameliorate”, “castigate”, “defenestrate”]
>>> numbers = [17, 123]
>>> empty = [ ]
>>> print (vocabulary, numbers, empty)
```

[‘ameliorate’, ‘castigate’, ‘defenestrate’] [17, 123][ ]

List

Like numeric 0 values and the empty string, the empty list is false in a Boolean expression.

```
>>> if [ ]:
    print ("This is true")
else:
    print ("This is false")


This is false
```

List - Accessing elements

The bracket operator selects a single item from a list (similar to string)

```
>>> numbers = [17, 123]
>>> print(numbers[0])
17
```

List

index

Python is zero index based

List - Accessing elements

Any integer expression can be used as an index

```
>>> numbers = [17, 123]
>>> numbers[9-8]
123
>>> numbers[1.0]
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    numbers[1.0]
TypeError: list indices must be integers, not float
```

List - Accessing elements

If you try to read or write an element that does not exist, you get a runtime error:

```
>>> numbers = [17, 123]
>>> numbers[2]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    numbers[2]
IndexError: list index out of range
```

List - Accessing elements

If an index has a negative value, it counts backward from the end of the list

```
>>> numbers[-1]
123
>>> numbers[-2]
17
>>> numbers[-3]
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    numbers[-3]
IndexError: list index out of range
```

List - Accessing elements

It is common to use a *loop variable* as a list index.

```
weekdays = ["monday", "tuesday", "wednesday", "thursday", "friday"]
i = 0
while i < 5:
    print (weekdays[i])
    i +=1
```

the body of the loop is only executed when *i* is 0, 1, 2, 3 and 4.

List - length

The function **len** returns the length of a list, which is equal to the number of its elements.

```
weekdays = ["monday", "tuesday", "wednesday", "thursday", "friday"]
i = 0
num = len(weekdays)
while i < num:
    print (weekdays[i])
    i +=1
```

List membership

in is a Boolean operator that tests membership in a sequence (similar to strings)

```
>>> 'friday' in weekdays
True
>>> 'sunday' in weekdays
False
>>>
```

List membership

the **not in** test whether an element is not a member of a list

```
>>> 'sunday' not in weekdays
True
>>>
```

List operations

The + operator concatenates lists

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> c = a + b
>>> print (c)
[1, 2, 3, 4, 5, 6]
```

List operations

the * operator repeats a list a given number of times

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1,2,3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Class Activity 1

Write a Python program to multiply all the items in a list.

List slices

The slice operations in strings also work similarly on lists

```
>>> a_list = ['a','b','c','d','e','f']
>>> a_list[1:3]
['b', 'c']
>>> a_list[3:]
['d', 'e', 'f']
>>> a_list[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

List - range function

The range function takes two arguments and returns a list that contains all the integers from the first to the second, including the first but not the second.

print(range(1,5))

List - range function

There are two other forms of range.

1. Single argument range
2. Triple argument range

```
>>> for i in range(10):
    print(i)
0
1
2
3
4
5
6
7
8
9
```

List - range function

There are two other forms of range.

1. Single argument range
2. Triple argument range

```
>>> for i in range(20,4,-5):
    print(i)
20
15
10
5
```

Lists are mutable

Unlike strings, lists are mutable, which means we can change their elements.

the bracket operator on the left side of an assignment is used to update a list

```
>>> fruit = ["banana", "apple", "quince"]
>>> fruit[0] = "pear"
>>> fruit[-1] = "orange"
>>> print(fruit)
['pear', 'apple', 'orange']
```

Lists are mutable

With the slice operator we can update several elements at once.

```
>>> a_list = ['a','b','c','d','e','f']
>>> a_list[1:3] = ['x','y']
>>> print(a_list)
['a', 'x', 'y', 'd', 'e', 'f']
```

Lists are mutable

We can also remove elements from a list by assigning the empty list to them

```
>>> a_list = ['a','b','c','d','e','f']

>>> a_list[1:3] = []

>>> print(a_list)

['a', 'd', 'e', 'f']
```

Lists are mutable

we can add elements to a list by squeezing them into an empty slice at the desired location

```
>>> a_list = ['a','d','f']
>>> a_list[1:1] = ['b','c']
>>> print(a_list)
['a', 'b', 'c', 'd', 'f']
>>> a_list[4:4] = ['e']
>>> print(a_list)
['a', 'b', 'c', 'd', 'e', 'f']
```

List deletion

del removes an element from a list:

```
>>> a = ['one','two','three']
>>> del a[1]
>>> a
['one', 'three']
```

del handles negative indices and causes a runtime error if the index is out of range.

List - Objects and values

Since strings are immutable, Python optimizes resources by making two names that refer to the same string value refer to the same object.

```
>>> a = "banana"
>>> b = "banana“
>>> a == b
True
>>> a is b
True
```

List - Objects and values

Lists are mutable, hence

```
>>> a = [1,2]
>>> b = [1,2]
>>> a == b
True
>>> a is b
False
```

List - Aliasing

Since variables refer to objects, if we assign one variable to another, both variables refer to the same object

```
>>> a = [1,2,3]
>>> b = a
>>> a is b
True
```

List - Aliasing

Because the same list has two different names, a and b, we say that it is **aliased**. Changes made with one alias affect the other

```
>>> b[0] = 5
>>> print(a)
[5, 2, 3]
```

List - Cloning

Cloning enables us to modify a list and also keep a copy of the original

```
>>> a = [1,2,3]
>>> b = a[:]
>>> print(b)
[1, 2, 3]
```

b[0] = 5
Print a and b

Class Activity 2

Write a Python program to clone or copy a list and empty the original list.

List – for loop

The generalized syntax of a for loop is:

```
for VARIABLE in LIST:
    BODY
```

This statement is equivalent to:

```
i = 0
while i < len(LIST):
    VARIABLE = LIST[i]
    BODY
    i += 1
```

List – for loop

```
for weekday in weekdays:
  print (weekday)

for number in range(20):
  if number % 3 == 0:
    print (number)

for fruit in [“banana”, “apple”, “quince”]:
  print (“I like to eat ” + fruit + “s!”)
```

The for loop is more concise because we can eliminate the loop variable, i.

List – for loop

Modifying each element while traversing a list.

```
numbers = [1,2,3,4,5]
for index in range(len(numbers)):
    numbers[index] = numbers[index]**2
```

List – for loop

Enumerate generates both the ***index*** and the ***value*** associated with it during the list traversal.

```
numbers = [1,2,3,4,5]
for index, value in enumerate(numbers):
    numbers[index] = value**2
```

List – for loop

Implement the code below

```
for index, value in
enumerate(['banana','apple','pear','quince'])
  print (index, value)
```

Class Activity 3

Write a loop that traverses:
['spam!', 1, ['Brie', 'Roquefort', 'Pol le Veq'], [1, 2, 3]]
(b) Print the length of each element. What happens if you send an integer to len?
(c) Change 1 to 'one' and run your solution again.

List parameters

Passing a list as an argument actually passes a reference to the list not a copy of the list.

Since lists are mutable changes made to the parameter change the argument as well.

parameter

```
def double_stuff(a_list):
  for index, value in enumerate(a_list):
    a_list[index] = 2 * value
things = [2, 5, 'spam', 9.5]
double_stuff(things)
print(things)
[4, 10, 'SpamSpam', 19.0]
```

argument

List parameters

```
def double_stuff(a_list):
  new_list = [ ]
  for value in a_list:
    new_list += [2 * value]
  return new_list

things = [2, 5, 'spam', 9.5]
print(double_stuff(things))
[4, 10, 'SpamSpam', 19.0]

print(things)
[2, 5, 'spam', 9.5]
```

Functions which take lists as arguments and change them during execution are called **modifiers** and the changes they make are called **side effects.**

A **pure function** does not produce side effects.

Nested lists

Nested lists are often used to represent matrices.
For example, the matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
>>> matrix = [[1,2,3],[4,5,6],[7,8,9]]
```

Nested lists

matrix is a list with three elements, where each element is a row of the matrix.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
>>> matrix [1]
[4, 5, 6]
# Extract a single element
>>> matrix[1][1]
5
```

Strings and lists

List( ) takes a sequence type as an argument and creates a list out of its elements

```
>>> list("crunchy Frog")
['c', 'r', 'u', 'n', 'c', 'h', 'y', ' ', 'F', 'r', 'o', 'g']
```

List Methods

Method	Description
List(s)	Converts *s* to a list
s.append(x)	Appends a new element *x* to the end of *s*
s.extend(t)	Appends a new list *t* to the end of *s*
s.count(x)	Count occurrences of *x* in s
s.insert(i,x)	Inserts x at index i
s.pop([i])	Returns the element I and removes it from the list
s.remove(x)	Searches for x and removes it from s
s.reverse()	Reverses items of s in place
s.sort([key [,reverse]])	Sorts items of s in place

Python Programming

Tuple

Tuples and mutability

A tuple is an immutable sequence of items of any type. Unlike lists that are mutable

Syntactically, a tuple is a comma-separated sequence of values:

```
>>> tup = 2, 4, 6, 8, 10
```

It is conventional to enclose tuples in parentheses:

```
>>> tup = (2, 4, 6, 8, 10)
```

Tuples and mutability

To create a tuple with a single element, we have to include the final comma

```
>>> tup = (5,)
>>> type(tup)
<class 'tuple'>
```

Without the comma, Python treats (5) as an integer in parentheses:

```
>>> tup = (5)
>>> type(tup)
<class 'int'>
```

Tuples and mutability

Tuples support the same sequence operations as strings and lists. The index operator selects an element from a tuple.

```
>>> tup = ('a','b','c','d','e')
>>> tup[0]
'a'
```

And the slice operator selects a range of elements.

```
>>> tup[1:3]
('b', 'c')
```

Tuples and mutability

if we try to use item assignment to modify one of the elements of the tuple, we get an error

```
>>> tup[0] = 'x'

Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    tup[0] = 'x'
TypeError: 'tuple' object does not support item assignment
```

Tuples and mutability

We can modify the content of a tuple by first convert it to a list, modify it and convert it back to tuple

```
>>> tup = ('X','b','c','d','e')
>>> tup = list(tup)
>>> tup
['X', 'b', 'c', 'd', 'e']
>>> tup[0] = 'a'
>>> tup = tuple(tup)
>>> tup
('a', 'b', 'c', 'd', 'e')
>>>
```

Tuples assignment

It is useful to swap the values of two variables. In this case, we have to use a temporary variable.

```
a = 3
b = 4
>>> temp = a
>>> a = b
>>> b = temp
```

This approach is a bit *cumbersome*

Tuples assignment

tuple assignment solves this problem neatly

The left side is a tuple of **variables**; the right side is a tuple of **values**. Each value is assigned to its respective variable.

```
a = 3
b = 4
>>> a, b= b, a
print(a,b)
```

Tuples assignment

Naturally, the number of variables on the left and the number of values on the right have to be the same

```
>>> a,b,c,d = 1,2,3
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    a,b,c,d = 1,2,3
ValueError: need more than 3 values to unpack
```

Operations Applicable to **Mutable** Sequences

Method	Description
s[i] = v	Item assignment
s.[i:j] = t	Slice assignment
del s[i]	Item deletion
del s[i:j]	Slice deletion

Operations Applicable to **all** Sequences

Method	Description
s[i]	Returns element I of a sequence
s[i:j]	Returns a slice
len(s)	Number of element in s
min(s)	Minimum value in s
max(s)	Maximum value in s
Sum(s [,initial])	Sum of items in s
all(s)	Checks whether all items in s are True
any(s)	Checks whether all items in s are False

Python Programming

Tutorials

Exercise 1:

Write a Python program to get the largest number from a list.

Exercise 2:

Write a Python program to find the list of words that are longer than n from a given list of words.

Exercise 3:

Write a Python program to check a list is empty or not.

Exercise 4:

Write a Python program to generate and print a list except for the first 5 elements, where the values are square of numbers between 1 and 20 (both included).

Exercise 5:

Write a Python program to append a list to the second list.

Exercise 6:

Write a Python program to find the second smallest number in a list.

Exercise 7:

Write a Python program to print a specified list after removing the 0th, 4th and 5th elements.

Exercise 8:

Write a Python program to append a list to the second list.

Exercise 9:

Write a Python program to select an item randomly from a list.

Exercise 10:

Write a Python program to get the frequency of the elements in a list.

Exercise 11:

Write a Python program to remove duplicates from a list of lists.

Exercise 12:

Write a Python program to generate all sublists of a list.

Exercise 13:

Write a Python program to add an item in a tuple

Exercise 14:

Write a Python program to remove an item from a tuple

Next Lecture ...

Python Programming

Day 5: Sets and Dictionary