CORE PYTHON PROGRAMMING (WORKSHOP)

Statistical methods using Python Programming Agenda

Introduction to Python Programming, Input, Processing and Output

Displaying Output with the Print Function,

Comments, Variables, Reading Input from the Keyboard, Performing Calculations Operators. Type conversions, Expressions, More about Data Output.

Decision Structures and Boolean Logic: if, if-else, if-elif-else Statements, Nested Decision Structures, Comparing Strings, Logical Operators, Boolean Variables

Repetition Structures: recursion and non recursion, while loop, for loop, Calculating a Running Total, Input Validation Loops, Nested Loops

Python-syntax, statements, functions, Built-in-functions and Methods, Modules in python, Exception Handling.

Generating Random Numbers, Writing Our Own Value-Returning Functions, The math Module, Storing Functions in Modules

File and Exceptions: Introduction to File Input and Output, Using Loops to Process Files, Processing Records, Exceptions

Finding Items in Lists with in-Operator, List Methods and Useful Built-in Functions, Copying Lists, Processing Lists, Two-Dimensional Lists, Tuples. Strings: Basic String Operations, String Slicing, Testing, Searching, and Manipulating Strings.

Strings: Basic String Operations, String Slicing, Testing, Searching, and Manipulating Strings.

STRINGS

Python Strings

A string is a sequence of characters.

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

IndexError Exceptions :An IndexError exception will occur if you try to use an index that is out of range for a particular string

String Concatenation:common operation that performed on strings is concatenation, or appending one string to the end of another string.

Python Strings

Python provides several ways to access the individual characters in a string. Strings also have methods that allow you to perform operations on them.

Indexing :you can access the individual characters in a string is with an index.

How to create a string in Python?

Strings can be created by enclosing characters inside a single quote or double quotes.

Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

```
# 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)
```

How to access characters in a string?

The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator (colon).

Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

```
#How to access characters in a string?
str = 'programiz'
print('str = ', str)

#first character
print('str[0] = ', str[0])

#last character
print('str[-1] = ', str[-1])

#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

How to change or delete a string?

Strings are immutable

Elements of a string cannot be changed once it has been assigned.

simply reassign different strings to the same name

Concatenation of Two or More Strings

Joining of two or more strings into a single one is called concatenation.

The **+** operator does this in Python. Simply writing two string literals together also concatenates them.

***** operator can be used to repeat the string for a given number of times.

```
#Joining of two or more strings into a single one is called concatenation.

str1 = 'Hello'
str2 ='World!'

# using +
print('str1 + str2 = ', str1 + str2)

# using *
print('str1 * 3 =', str1 * 3)
```

Python 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 working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

```
str = 'Hello World!'

print str          # Prints complete string

print str[0]       # Prints first character of the string

print str[2:5]     # Prints characters starting from 3rd to 5th

print str[2:]      # Prints string starting from 3rd character

print str * 2      # Prints string two times

print str + "TEST" # Prints concatenated string
```

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

Complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x is the real part and b is the imaginary part of the complex number.

Python Strings

```
#Creating Strings
s1=str() #Create an empty string object
print("Empty String",s1)
s2=str("Welcome") #create a string object for Welcome
print("string object is",s2)
```

```
#String Concatenation
message = 'Hello ' + 'world'
print(message)
```

```
#Creating Strings
s1=str() #Create an empty string object
print("Empty String",s1)
s2=str("Welcome") #create a string object for Welcome
print("string object is",s2)
```

```
#Index Operator
var1 = 'Hello World!'
var2 = "Python Programming"
print("var1[0]: ", var1[0])
print("var2[1:5]: ", var2[1:5])
```

```
# appears in a string.
def main():
    # Create a variable to use to hold the count.
    # The variable must start with 0.
    count = 0
    my_string = input('Enter a sentence: ')
    # Count the Ts.
    for ch in my_string:
            if ch == 'T' or ch == 't':
                count += 1
        # Print the result.
    print('The letter T appears', count, 'times.')
# Call the main function.
main()
```

Python Strings

```
# 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)
```

Python Strings - How to access characters in a string?

```
#How to access characters in a string?
str = 'programiz'
print('str = ', str)

#first character
print('str[0] = ', str[0])

#last character
print('str[-1] = ', str[-1])

#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

Python Strings - String Manipulation

```
str1 = 'Hello'
str2 ='World!'

# using +
print('str1 + str2 = ', str1 + str2)

# using *
print('str1 * 3 =', str1 * 3)
```

Python Strings - Creating Strings

```
#Creating Strings
s1=str() #Create an empty string object
print("Empty String",s1)
s2=str("Welcome") #create a string object for Welcome
print("string object is",s2)
```

Python Strings - Index Operator

```
#Index Operator
var1 = 'Hello World!'
var2 = "Python Programming"
print("var1[0]: ", var1[0])
print("var2[1:5]: ", var2[1:5])
```

Python Strings – counts the number of times the letter T (uppercase or lowercase)

```
# This program counts the number of times the letter T (uppercase or lowercase)
# appears in a string.
def main():
    # Create a variable to use to hold the count.
    # The variable must start with 0.
    count = 0
    # Get a string from the user.name = 'Juliet'
    for ch in name:
        ch = 'X'
        print(name)
    my_string = input('Enter a sentence: ')
    # Count the Ts.
    for ch in my_string:
            if ch == 'T' or ch == 't':
                count += 1
        # Print the result.
    print('The letter T appears', count, 'times.')
# Call the main function.
main()
```

Python Strings –
Demonstrates the repetition operator.

```
# This program demonstrates the repetition operator.
def main():
    # Print nine rows increasing in length.
    for count in range(1, 10):
        print('Z' * count)
    # Print nine rows decreasing in length.
    for count in range(8, 0, -1):
        print('Z' * count)
# Call the main function.
main()
```

Python Strings

Some string testing methods

Metho d	Descripti on
isalnum()	Returns true if the string contains only alphabetic letters or digits and is at least one character in length. Returns false otherwise.
isalpha()	Returns true if the string contains only alphabetic letters and is at least one character in length. Returns false otherwise
isdigit()	Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.
islower()	Returns true if all of the alphabetic letters in the string are lowercase, and the string contains at least one alphabetic letter. Returns false otherwise.
isspace()	Returns true if the string contains only whitespace characters and is at least one character in length. Returns false otherwise. (Whitespace characters arespaces, newlines (\n), and tabs (\t).
isupper()	Returns true if all of the alphabetic letters in the string are uppercase, and the string contains at least one alphabetic letter. Returns false otherwise.

Python Strings –
Demonstrates several string testing methods.

```
def main():
    # Get a string from the user.
    user_string = input('Enter a string: ')
    print('This is what I found about that string:')
    # Test the string.
    if user_string.isalnum():
        print('The string is alphanumeric.')
    if user_string.isdigit():
        print('The string contains only digits.')
    if user_string.isalpha():
        print('The string contains only alphabetic characters.')
    if user_string.isspace():
        print('The string contains only whitespace characters.')
    if user_string.islower():
        print('The letters in the string are all lowercase.')
    if user_string.isupper():
        print('The letters in the string are all uppercase.')
# Call the string.
main()
```

Python Strings –
String Testing Methods

```
#String Testing Methods
#isdigit method returns true if the string contains only numeric digits
string1 = '1200'
if string1.isdigit():
    print(string1, 'contains only digits.')
else:
    print(string1, 'contains characters other than digits.')
```

```
#String Testing Methods
#isdigit method returns true if the string contains only numeric digits
string2 = '123abc'
if string2.isdigit():
    print(string2, 'contains only digits.')
else:
    print(string2, 'contains characters other than digits.')
```

Python Strings

String Modification Methods

Method	Description
lower()	Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, or is not an alphabetic letter, is unchanged.
lstrip()	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the beginning of the string.
lstrip(char)	The char argument is a string containing a character. Returns a copy of the string with all instances of char that appear at the beginning of the string removed.
rstrip()	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the end of the string.
rstrip(char)	The char argument is a string containing a character. The method returns a copy of the string with all instances of char that appear at the end of the string removed.

Python Strings

Some string testing methods

```
string = "Hello World"

print(string.upper())

print(string.lower())

print(string.title())

print(string.capitalize())
print(string.swapcase())
```

```
string = "Hello World"

print(' '.join(reversed(string)))
```

```
word = "Hello World"

print(word.isalnum())        #check if all char are alphanumeric
print(word.isalpha())        #check if all char in the string are
alphabetic
print(word.isdigit())        #test if string contains digits
print(word.istitle())        #test if string contains title words
print(word.isupper())        #test if string contains upper case
print(word.islower())        #test if string contains lower case
print(word.isspace())        #test if string contains spaces
print(word.endswith('d'))    #test if string ends with a d
print(word.startswith('H'))  #test if string startswith H
```

Python Strings
Splitting a String

```
#Splitting a String
# This program demonstrates the split method.
def main():
    # Create a string with multiple words.
    my_string = 'One two three four'
    # Split the string.
    word_list = my_string.split()
    # Print the list of words.
    print(word_list)
# Call the main function.
main()
```

```
letters = 'abcd'
print(letters, letters.upper())
```

```
again = 'y'
while again.lower() == 'y':
    print('Hello')
    print('Do you want to see that again?')
    again = input('y = yes, anything else = no: ')
```

Python Strings

Splitting a String

```
# This program calls the split method, using the # '/' character as a separator.
def main():
    # Create a string with a date.
    date_string = '11/26/2014'
    # Split the date.
    date_list = date_string.split('/')
    # Display each piece of the date.
    print('Month:', date_list[0])
    print('Day:', date_list[1])
    print('Year:', date_list[2])
    # Call the main function.
main()
```

CONTAINERS

Python Containers

Python includes several built-in container types:

- Lists
- dictionaries
- sets, and
- tuples.

INTRODUCING LISTS

Python Lists

Sequences

A sequence is an object that holds multiple items of data, stored one after the other. You can perform operations on a sequence to examine and manipulate the items stored in it.

Python Lists

A list is an object that contains multiple data items.

Lists are mutable, which means that their contents can be changed during a program’s execution.

Lists are dynamic data structures, meaning that items may be added to them or removed from them.

You can use indexing, slicing, and various methods to work with lists in a program

If a Python list were like a pencil, then a Python Tuple would be like a pen.

If you went for a walk, you could note your coordinates at any instant in an (x,y) tuple.

If you wanted to record your journey, you could append your location every few seconds to a list.

But you couldn't do it the other way around.

Python Lists

Lists are the most versatile of Python's compound data types.

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

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 indexes starting at 0 in the beginning of the list and working their way to end -1.

The plus (+) sign is the list concatenation operator

asterisk (*) is the repetition operator.

Python Lists

list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

You can perform operations on a sequence to examine and manipulate the items stored in it.

```
# empty list
my_list = []
print(my_list)
```

```
# list of integers
my_listintegers = [1, 2, 3]
print(my_listintegers)
```

```
# list with mixed datatypes
my_listmixed = [1, "Hello", 3.4]
print(my_listmixed )
```

```
# nested list
my_listnested = ["mouse", [8, 4, 6], ['a']]
print(my_listnested)
```

Python Lists

```
list1=list() #creating an empty list
print("list1 = ",list1)
```

```
list2=list([2,3,4]) #creating an list with elements 2,3,4
print("list2 = ",list2)
```

```
list3=list(["red","blue","green"]) #creating an list with srings
print("list3 = ",list3)
```

```
list4=list(range(2,10)) #creating an list with elements 2,3,4
print("list4 = ",list4)
```

```
list5=list("red") #creating an list with characters
print("list5 = ",list5)
```

```
list6=list([9,"Ten",11,"Twelve"]) #creating an list with Mixed type
print("list6 = ",list6)
```

Python Lists

List can contain the elements of the same type or mixed type

List is a sequence type

Functions for lists

```
list7=[1,2,3,4,5,56,78,36,88]
print("list7 = ",list7)
list8=len(list7)
print("list8 = ",list8)
list9=max(list7)
print("list9 = ",list9)
list10=sum(list7)
print("list10 = ",list10)
list11=2 in list7
print("list11 = ",list11)
list12=88 not in list7
print("list12 = ",list12)
```

List comprehensions:

When programming, frequently we want to transform one type of data into another. As a simple example, consider the following code that computes square numbers:

```
# This program display and sum of a list of
numbers using While loop
list=[10,20,30,40,50]
sum=0
i=0
while I < len(list):
    print(list[i] ) # display the element from list
    sum+=list[i]  #  Add each element to sum
    I+=1
print(“Sum = “,sum)
```

List comprehensions:

When programming, frequently we want to transform one type of data into another. As a simple example, consider the following code that computes square numbers:

simpler using a list comprehension

```
#Simpler List comprehensions:
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Prints [0, 1, 4, 9, 16]
```

List comprehensions:

When programming, frequently we want to transform one type of data into another. As a simple example, consider the following code that computes square numbers:

simpler using a list comprehension

```
#List comprehensions can also contain conditions:
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Prints "[0, 4, 16]"
```

Repetition Structures

The for Loop: A Count-Controlled Loop

```
# This program display and sum of a list of numbers usig While loop
list=[10,20,30,40,50]
sum=0
i=0
while I < len(list):
        print(list[i] ) # display the element from list
        sum+=list[i]  #  Add each element to sum
        I+=1
print(“Sum = “,sum)
```

How to access elements from a list?

List Index

```
my_list = ['p','r','o','b','e']
# Output: p
print(my_list[0])

# Output: o
print(my_list[2])

# Output: e
print(my_list[4])

# Error! Only integer can be used for indexing
# my_list[4.0]

# Nested List
n_list = ["Happy", [2,0,1,5]]

# Nested indexing

# Output: a
print(n_list[0][1])

# Output: 5
print(n_list[1][3])
```

How to access elements from a list?

Negative indexing

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

```
my_list = ['p','r','o','b','e']

# Output: e
print(my_list[-1])

# Output: p
print(my_list[-5])
```

How to slice lists in Python?

a range of items in a list by using the slicing operator (colon).

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

```
#slice lists in Python
my_list = ['p','r','o','g','r','a','m','i','z']
# elements 3rd to 5th
print(my_list[2:5])

# elements beginning to 4th
print(my_list[:-5])

# elements 6th to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

## How to change or add elements to a list?

List are mutable, meaning, their elements can be changed unlike string or tuple.

assignment operator (=) to change an item or a range of items.

```
#How to change or add elements to a list?
# mistake values
odd = [2, 4, 6, 8]

# change the 1st item
odd[0] = 1

# Output: [1, 4, 6, 8]
print(odd)

# change 2nd to 4th items
odd[1:4] = [3, 5, 7]

# Output: [1, 3, 5, 7]
print(odd)
```

How to change or add elements to a list?

List are mutable, meaning, their elements can be changed unlike string or tuple.

assignment operator (=) to change an item or a range of items.

We can add one item to a list using append() method or add several items using extend() method.

We can add one item to a list using append() method or add several items using extend() method.

```
odd = [1, 3, 5]

odd.append(7)

# Output: [1, 3, 5, 7]
print(odd)

odd.extend([9, 11, 13])

# Output: [1, 3, 5, 7, 9, 11, 13]
print(odd)
```

How to change or add elements to a list?

use + operator to combine two lists. This is also called concatenation..

* operator repeats a list for the given number of times.

We can add one item to a list using append() method or add several items using extend() method.

```
#   + operator to combine two lists. This is also called concatenation.
# * operator repeats a list for the given number of times.
odd = [1, 3, 5]

# Output: [1, 3, 5, 9, 7, 5]
print(odd + [9, 7, 5])

#Output: ["re", "re", "re"]
print(["re"] * 3)
```

How to change or add elements to a list?

Insert one item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.

We can add one item to a list using append() method or add several items using extend() method.

```
# insert one item at a desired location by using the method insert() or
#insert multiple items by squeezing it into an empty slice of a list.
odd = [1, 9]
odd.insert(1,3)

# Output: [1, 3, 9]
print(odd)

odd[2:2] = [5, 7]

# Output: [1, 3, 5, 7, 9]
print(odd)
```

## How to delete or remove elements from a list?

Insert one item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.

```
#How to delete or remove elements from a list?
#delete one or more items from a list using the keyword del. It can even
delete the list entirely.
my_list = ['p','r','o','b','l','e','m']

# delete one item
del my_list[2]
# Output: ['p', 'r', 'b', 'l', 'e', 'm']
print(my_list)
# delete multiple items
del my_list[1:5]
# Output: ['p', 'm']
print(my_list)
# delete entire list
del my_list
# Error: List not defined
print(my_list)
```

How to delete or remove elements from a list?

remove() method to remove the given item or pop() method to remove an item at the given index.

```
#remove() method to remove the given item or pop() method to remove an
item at the given index.
#pop() method removes and returns the last item if index is not provided.
#This helps us implement lists as stacks (first in, last out data structure).
#use the clear() method to empty a list.
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')
# Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list)
# Output: 'o'
print(my_list.pop(1))
# Output: ['r', 'b', 'l', 'e', 'm']
print(my_list)
# Output: 'm'
print(my_list.pop())
# Output: ['r', 'b', 'l', 'e']
print(my_list)
my_list.clear()
# Output: []
print(my_list)
```

List Membership Test

test if an item exists in a list or not, using the keyword in.

```
#test if an item exists in a list or not, using the keyword in.
my_list = ['p','r','o','b','l','e','m']

# Output: True
print('p' in my_list)

# Output: False
print('a' in my_list)

# Output: True
print('c' not in my_list)
```

Iterating Through a List

Using a for loop we can iterate though each item in a list.

```
#Using a for loop we can iterate though each item in a list.
for fruit in ['apple','banana','mango']:
    print("I like",fruit)
```

Python Lists

Using List Elements in a Math Expression

Megan owns a small neighborhood coffee shop, and she has six employees who work as baristas (coffee bartenders). All of the employees have the same hourly pay rate.

Megan has asked you to design a program that will allow her to enter the number of hours worked by each employee and then display the amounts of all the employees' gross pay.

You determine that the program should perform the following steps:

1. For each employee: get the number of hours worked and store it in a list element.
2. For each list element: use the value stored in the element to calculate an employee's gross pay. Display the amount of the gross pay.

Python Lists

Example of how user input can be assigned to the elements of a list.
This program gets sales amounts from the user and assigns them to a list.

```
# This program calculates the gross pay for
# each of Megan's baristas.
# NUM_EMPLOYEES is used as a constant for the
# size of the list.
NUM_EMPLOYEES = 6
def main():
    # Create a list to hold employee hours.
    hours = [0] * NUM_EMPLOYEES
    # Get each employee's hours worked.
    for index in range(NUM_EMPLOYEES):
        print('Enter the hours worked by employee ', index + 1, ': ', sep='', end='')
        hours[index] = float(input())
        # Get the hourly pay rate.
        pay_rate = float(input('Enter the hourly pay rate: '))
        # Display each employee's gross pay.
    for index in range(NUM_EMPLOYEES):
        gross_pay = hours[index] * pay_rate
        print('Gross pay for employee ', index + 1, ': $', \
        format(gross_pay, ',.2f'), sep='')
# Call the main function.
main()
```

Python Lists

What will the following code display?

```
numbers = [1, 2, 3, 4, 5]
numbers[2] = 99
print(numbers)
```

What will the following code display?

```
numbers = list(range(3))
print(numbers)
```

What will the following code display?

```
numbers = [10] * 5
print(numbers)
```

What will the following code display?

```
numbers = list(range(1, 10, 2))
for n in numbers:
print(n)
```

Python Lists

What will the following code display?

```
numbers = [1, 2, 3, 4, 5]
print(numbers[-2])
```

What will the following code display?

```
numbers1 = [1, 2, 3]
numbers2 = [10, 20, 30]
numbers3 = numbers1 + numbers2
print(numbers1)
print(numbers2)
print(numbers3)
```

What will the following code display?

```
numbers1 = [1, 2, 3]
numbers2 = [10, 20, 30]
numbers2 += numbers1
print(numbers1)
print(numbers2)
```

Python Lists

Concatenating Lists

Concatenate means to join two things together. You can use the + operator to concatenate two lists.

```
#Concatenating Lists
#Keep in mind that you can concatenate lists only with other lists. If you try
to
#concatenate a list with something that is not a list, an exception will be
raised.
list1 = [1, 2, 3, 4]
print("list1 = ",list1)
list2 = [5, 6, 7, 8]
print("list2 = ",list2)
list3 = list1 + list2
```

```
girl_names = ['Joanne', 'Karen', 'Lori']
print("girl_names = ",girl_names)
boy_names = ['Chris', 'Jerry', 'Will']
print("boy_names = ",boy_names)
all_names = girl_names + boy_names
print("all_names = ",all_names)
```

Keep in mind that you can concatenate lists only with other lists.

If you try to concatenate a list with something that is not a list, an exception will be raised.

Python Lists

List Slicing

A slicing expression selects a range of elements from a sequence.

A *slice* is a span of items that are taken from a sequence

When you take a slice from a list, you get a span of elements from within the list.

To get a slice of a list, you write an expression in the following general format:

list_name[*start* : *end*]

Starting Index and Ending index may be omitted

Slice is a sublist from index start to index end

Python Lists

List Slicing

```
numbers = [1, 2, 3, 4, 5]
print(numbers)
print(numbers[1:3])
```

```
numbers = [1, 2, 3, 4, 5]
print(numbers)
print(numbers[:3])
```

```
numbers = [1, 2, 3, 4, 5]
print(numbers)
print(numbers[2:])
```

```
#leave out both the start and end index in a slicing expression, you get a copy of the entire list
numbers = [1, 2, 3, 4, 5]
print(numbers)
print(numbers[:])
```

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(numbers)
print(numbers[1:8:2])   #third number inside the brackets is the step value
```

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(numbers)
print(numbers[-5:])
```

Negative numbers as indexes in slicing expressions to reference positions relative to the end of the list.

Python adds a negative index to the length of a list to get the position referenced by that index.

Python Lists

List Slicing

```
numbers = [1, 2, 3, 4,
5]
print(numbers)
print(numbers[1:3])
```

```
numbers = [1, 2, 3, 4,
5]
print(numbers)
print(numbers[:3])
```

```
numbers = [1, 2, 3, 4,
5]
print(numbers)
print(numbers[2:])
```

```
#leave out both the start and end index in a slicing expression, you get a
copy of the entire list
numbers = [1, 2, 3, 4, 5]
print(numbers)
print(numbers[:])
```

```
#List Slicing
#A slicing expression selects a range of elements from a sequence.

days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday’]

mid_days = days[2:5]

print(mid_days)
```

Python Lists

List Slicing

```
#leave out both the start and end index in a slicing expression, you get a
copy of the entire list
numbers = [1, 2, 3, 4, 5]
print(numbers)
print(numbers[:])
```

```
#List Slicing
#A slicing expression selects a range of elements from a sequence.

days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday’]

mid_days = days[2:5]

print(mid_days)
```

Python Lists

List Slicing

Invalid indexes do not cause slicing expressions to raise an exception.
For example:

If the *end* index specifies a position beyond the end of the list, Python will use the length of the list instead.

If the *start* index specifies a position before the beginning of the list, Python will use 0 instead.

If the *start* index is greater than the *end* index, the slicing expression will return an empty list.

Finding Items in Lists with the in Operator

You can search for an item in a list using the in operator.

In Python you can use the in operator to determine whether an item is contained in a list.

```
# This program demonstrates the in operator used with a list.
def main():
    # Create a list of product numbers.
    prod_nums = ['V475', 'F987', 'Q143', 'R688']
    # Get a product number to search for.
    search = input('Enter a product number: ')
    # Determine whether the product number is in the list.
    if search in prod_nums:
        print(search, 'was found in the list.')
    else:
        print(search, 'was not found in the list.')
# Call the main function.
main()
```

List Methods and Useful Built-in Functions

Lists have numerous methods that allow you to work with the elements that they contain.

Python also provides some built-in functions that are useful for working with lists.

```
# This program demonstrates the in operator used with a list.
def main():
    # Create a list of product numbers.
    prod_nums = ['V475', 'F987', 'Q143', 'R688']
    # Get a product number to search for.
    search = input('Enter a product number: ')
    # Determine whether the product number is in the list.
    if search in prod_nums:
        print(search, 'was found in the list.')
    else:
        print(search, 'was not found in the list.')
# Call the main function.
main()
```

List Methods and Useful Built-in Functions

A **method** is a function that takes a class instance as its first parameter.

Methods are members of classes.

List Methods and Useful Built-in Functions

A few of the list methods

Metho d	Descripti on
append(item)	Adds *item* to the end of the list.
index(item)	Returns the index of the first element whose value is equal to item. A ValueError exception is raised if item is not found in the list.
insert(index, item)	Inserts item into the list at the specified index.

When an item is inserted into a list, the list is expanded in size to accommodate the new item.

The item that was previously at the specified index, and all the items after it, are shifted by one position toward the end of the list.

No exceptions will occur if you specify an invalid index

If you specify an index beyond the end of the list, the item will be added to the end of the list

If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.

List Methods and Useful Built-in Functions

A few of the list methods

Metho d	Descripti on
sort()	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value).
remove(item)	Removes the first occurrence of item from the list. A ValueError exception is raised if item is not found in the list.+
reverse()	Reverses the order of the items in the list.

List Methods and Useful Built-in Functions

A few of the list methods

```
# Adds List Element as value of List.
List = ['Mathematics', 'chemistry', 1997, 2000]
List.append(20544)
print(List)
```

```
List = [1, 2, 3, 4, 5]
print(sum(List))
```

```
List = ['Mathematics', 'chemistry', 1997, 2000]
# Insert at index 2 value 10087
List.insert(2,10087)
print(List)
print(List[0])
```

```
List1 = [1, 2, 3]
List2 = [2, 3, 4, 5]

# Add List2 to List1
List1.extend(List2)
print(List1)

#Add List1 to List2 now
List2.extend(List1)
print(List2)
```

```
#count():Calculates total occurrence of given
element of List.
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.count(1))
```

List Methods and Useful Built-in Functions

A few of the list methods

```
#index(): Returns the index of first occurrence. Start and End index are not necessary parameters
#List.index(element[,start[,end]])  (Format)
.
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.index(2))
```

```
#index(): Returns the index of first occurrence. Start and End index are not necessary parameters.
#List.index(element[,start[,end]])  (Format)

List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.index(2))
```

List Methods and Useful Built-in Functions

Deletion of List Elements

To Delete one or more elements, i.e. remove an element, many built in functions can be used, such as pop() & remove() and keywords such as del.

pop(): Index is not a necessary parameter, if not mentioned takes the last index

list.pop([index])

Note: Index must be in range of the List, elsewise IndexErrors occurs.

```
#index(): Returns the index of first occurrence. Start and End index are not necessary parameters.
#List.index(element[,start[,end]])  (Format)

List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.index(2))
```

```
List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
print(List.pop(0))
```

List Methods and Useful Built-in Functions

Deletion of List Elements

```
#del() : Element to be deleted is mentioned using list name and index.
#Syntax:  del list.[index]
List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
del List[0]
print(List)
```

```
#remove(): Element to be deleted is mentioned using list name and element.
#Syntax:  list.remove(element)
List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
List.remove(3)
print(List)
```

List Methods and Useful Built-in Functions

Python Built-in Exceptions

Exception	Cause of Error
AirthmeticError	For errors in numeric calculation.
AssertionErrorggj	When an attribute assignment or the reference fails.
EOFError	If there is no input or the file pointer is at EOF.
Exception	For errors that occur outside the Python environment.
EnvironmentError	If the assert statement fails.
FloatingPointError	It occurs when the floating point operation fails.
GeneratorExit	If a generator’s <close()> method gets called.

List Methods and Useful Built-in Functions

Python Built-in Exceptions

Exception	Cause of Error
AirthmeticEr ror	For errors in numeric calculation.
AssertionError ggj	When an attribute assignment or the reference fails.
EOFError	If there is no input or the file pointer is at EOF.
Exception	For errors that occur outside the Python environment.
EnvironmentE rror	If the assert statement fails.
FloatingPointE rror	It occurs when the floating point operation fails.
GeneratorExit	If a generator’s <close()> method gets called.

List Methods and Useful Built-in Functions

Python Built-in Exceptions

Exception	Cause of Error
ImportError	It occurs when the imported module is not available.
IOError	If an input/output operation fails.
IndexError	When the index of a sequence is out of range.
KeyError	If the specified key is not available in the dictionary.
KeyboardInterrupt	When the user hits an interrupt key (Ctrl+c or delete).
MemoryError	If an operation runs out of memory.
NameError	When a variable is not available in local or global scope.

List Methods and Useful Built-in Functions

Python Built-in Exceptions

Exception	Cause of Error
NotImplementedError	If an abstract method isn’t available..
OSError	When a system operation fails.
OverflowError	It occurs if the result of an arithmetic operation exceeds the range.
ReferenceError	When a weak reference proxy accesses a garbage collected reference.
RuntimeError	If the generated error doesn’t fall under any category.
StandardError	It is a base class for all built-in exceptions except <StopIteration> and <SystemExit>.
StopIteration	The <next()> function has no further item to be returned.

List Methods and Useful Built-in Functions

Python Built-in Exceptions

Exception	Cause of Error
SyntaxError	It occurs if the indentation is not proper.
TabError	For inconsistent tabs and spaces.
SystemError	When interpreter detects an internal error.
SystemExit	The <sys.exit()> function raises it.
TypeError	When a function is using an object of the incorrect type.
UnboundLocalError	If the code using an unassigned reference gets executed.
UnicodeError	For a Unicode encoding or decoding error.
ValueError	When a function receives invalid values.
ZeroDivisionEr	If the second operand of division or modulo operation is zero.

List Methods and Useful Built-in Functions

```
list1=[2,3,4,1,32,4]
list1.append(19)
print(list1)

list1.count(4)
print(list1)

list2=[99,54]
list1.extend(list2)
print(list1)

a=list1.index(4)
print(a)

b=list1.insert(1,25)
print(b)

c=list1.pop(4)
print(c)

d=list1.remove(32)
print(d)
e=list1.reverse()

print(e)
f=list1.sort()
print(f)
```

List Methods and Useful Built-in Functions

```
#This program demonstrates how to get the index of an item in a list and then
#replace that item with a new item.
def main():
    # Create a list with some items.
    food = ['Pizza', 'Burgers', 'Chips']
    # Display the list.
    print('Here are the items in the food list:')
    print(food)
    # Get the item to change.
    item = input('Which item should I change? ')
    try:
        # Get the item's index in the list.
        item_index = food.index(item)
        # Get the value to replace it with.
        new_item = input('Enter the new value: ')
        # Replace the old item with the new item.
        food[item_index] = new_item
        # Display the list.
        print('Here is the revised list:')
        print(food)
    except ValueError:
        print('That item was not found in the list.')
# Call the main function.
main()
```

List Methods and Useful Built-in Functions

```
#The remove Method The remove method removes an item from the list.
# This program demonstrates how to use the remove
# method to remove an item from a list.
def main():
    # Create a list with some items.
    food = ['Pizza', 'Burgers', 'Chips']
    # Display the list.
    print('Here are the items in the food list:')
    print(food)
    # Get the item to change.
    item = input('Which item should I remove? ')
    try:
        # Remove the item.
        food.remove(item)
        # Display the list.
        print('Here is the revised list:')
        print(food)
    except ValueError:
        print('That item was not found in the list.')
# Call the main function.
main()
```

Totaling the Values in a List

```
# This program calculates the total of the values
# in a list.

def main():
    # Create a list.
    numbers = [2, 4, 6, 8, 10]
    # Create a variable to use as an accumulator.
    total = 0
    # Calculate the total of the list elements.
    for value in numbers:
        total += value
        # Display the total of the list elements.
        print('The total of the elements is', total)
# Call the main function.
main()
```

Using nested list comprehension

```
# In Python, we can implement a matrix as nested list (list inside a list).
#We can treat each element as a row of the matrix.

# Program to add two matrices
# using list comprehension

X = [[1,2,3],
    [4 ,5,6],
    [7 ,8,9]]

Y = [[9,8,7],
    [6,5,4],
    [3,2,1]]

result = [[X[i][j] + Y[i][j]  for j in range
(len(X[0]))] for i in range(len(X))]

for r in result:
    print(r)
```

Lists
Assignments

Megan owns a small neighborhood coffee shop, and she has six employees who work as baristas (coffee bartenders).

All of the employees have the same hourly pay rate.

Megan has asked you to design a program that will allow her to enter the number of hours worked by each employee and then display the amounts of all the employees' gross pay.

You determine that the program should perform the following steps:

1. For each employee: get the number of hours worked and store it in a list element.

2. For each list element: use the value stored in the element to calculate an employee's gross pay. Display the amount of the gross pay.

Lists

Assignment-solution   Contd....

```
#Using List Elements in a Math Expression
# This program calculates the gross pay for each of Megan's baristas.
# NUM_EMPLOYEES is used as a constant for the size of the list.
NUM_EMPLOYEES = 6
def main():
    # Create a list to hold employee hours.
    hours = [0] * NUM_EMPLOYEES
    # Get each employee's hours worked.
    for index in range(NUM_EMPLOYEES):
        print('Enter the hours worked by employee ', index + 1, ': ', sep='', end='')
        hours[index] = float(input())
        # Get the hourly pay rate.
        pay_rate = float(input('Enter the hourly pay rate: '))
        # Display each employee's gross pay.
    for index in range(NUM_EMPLOYEES):
        gross_pay = hours[index] * pay_rate
        print('Gross pay for employee ', index + 1, ': $', \
        format(gross_pay, ',.2f'), sep='')
# Call the main function.
main()
```

Lists

Assignments

Megan owns a small neighborhood coffee shop, and she has six employees who work as baristas (coffee bartenders).
All of the employees have the same hourly pay rate.
Megan has asked you to design a program that will allow her to enter the number of hours worked by each employee and then display the amounts of all the employees' gross pay.
You determine that the program should perform the following steps:
1. For each employee: get the number of hours worked and store it in a list element.
2. For each list element: use the value stored in the element to calculate an employee's gross pay. Display the amount of the gross pay.

Suppose Megan's business increases and she hires two additional baristas.
This would require you to change the program so it processes eight employees instead of six. Because you used a constant for the list size, this is a simple modification—you just change the statement in line 6 to read:
NUM_EMPLOYEES = 8

Two-Dimensional Lists

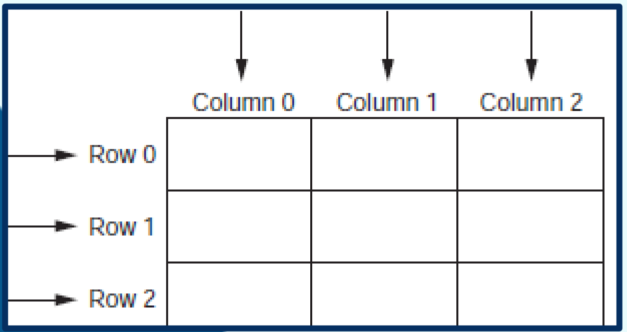
A two-dimensional list is a list that has other lists as its elements.

Data in a Table or a Matrix can be stored in a two dimensional list

Use a list to store two-dimensional data, such as a matrix or a Table.

Distance in Table(in miles)

	Chicago	Boston	Newyork	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1505	1723	1842
Newyork	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1505	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0



```
distances=[
          [0,983,787,714,1375,967,1087],
          [983,0,214,1102,1505,1723,1842],
          [787,214,0,888,1549,1548,1627],
          [714,1102,888,0,661,781,810],
          [1375,1505,1549,661,0,1426,1187],
          [967,1723,1548,781,1426,0,239],
          [1087,1842,1627,810,1187,239,0]
         ]
#Each element in the distances list is another list.
#distances are considered as nested list
#a two dimensional list is used tosstore two-dimensional
data
print(distances)
print(distances[1],[1])
print(distances[1][1])
```

Two-Dimensional Lists

A two-dimensional list is a list that has other lists as its elements.

```
# This program assigns random numbers to a two-dimensional list
import random
# Constants for rows and columns
ROWS = 3
COLS = 4

def main():
    # Create a two-dimensional list.
    values = [[0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0]]
    # Fill the list with random numbers.
    for r in range(ROWS):
        for c in range(COLS):
            values[r][c] = random.randint(1, 100)
    # Display the random numbers.
    print(values)
# Call the main function.
main()
```

Two-Dimensional Lists - ASSIGNMENT

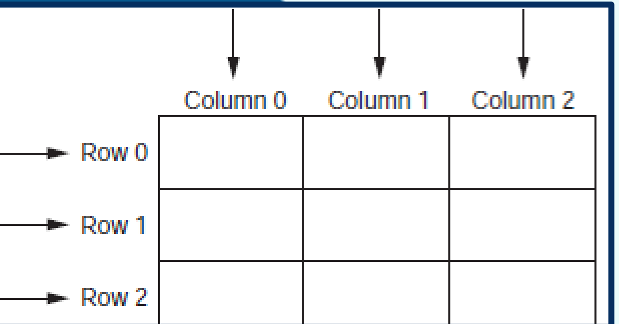
Write a program that grades multiple choice tests

Suppose there are eight students and ten questions and the answers are stored in a two dimensional list.

Each row records a student's answers to the questions as shown in the following

Students Answers to the Questions

	0	1	2	3	4	5	6	7	8	9
Student0	A	B	A	C	C	D	E	E	A	D
Student1	D	B	A	B	C	A	E	E	A	D
student2	E	D	D	A	C	B	E	E	A	D
Student3	C	B	A	E	D	C	E	E	A	D
Student4	A	B	D	C	C	D	E	E	A	D
Student5	B	B	E	C	C	D	E	E	A	D
Student6	B	B	A	C	C	D	E	E	A	D
Student7	E	B	E	C	C	D	E	E	A	D



The Key is stored in a one dimensional list

Key to the questions

Questions	0	1	2	3	4	5	6	7	8	9
Key	**D**	**B**	**D**	**C**	**C**	**D**	**A**	**E**	**A**	**D**

Two-Dimensional Lists

Write a program that grades multiple choice tests SOLUTION

```
def main():
    #Students answers to the questions
    Ans=[
             ["A","B","A","C","C","D","E","E","A","D"],
             ["D","B","A","B","C","A","E","E","A","D"],
             ["E","D","D","A","C","B","E","E","A","D"],
             ["C","B","A","E","D","C","E","E","A","D"],
             ["B","B","E","C","C","D","E","E","A","D"],
             ["B","B","A","C","C","D","E","E","A","D"],
             ["E","B","E","C","C","D","E","E","A","D"]
         ]
    #Key to the questions
    keys=["D","B","D","C","C","D","A","E","A","D"]

    print("Key to answers = ",keys)
    print("Student Answers = ", Ans)
    #Grade to all answers
    for i in range (len(Ans)):
        #Grade one sudent
        correctcount=0
        for j in range(len(Ans[i])):
            if Ans[i][j]==keys[j]:
                correctcount +=1
        print("student", i ," Answers correct count is " ,correctcount)
main()
```

Lists
Assignments

Processing a List
Dr. LaClaire gives a series of exams during the semester in her chemistry class. At the end of the semester she drops each student’s lowest test score before averaging the scores.

She has asked you to design a program that will read a student’s test scores as input and calculate the average with the lowest score dropped. Here is the algorithm that you developed:

1. *Get the student’s test scores.*
2. *Calculate the total of the scores.*
3. *Find the lowest score.*
4. *Subtract the lowest score from the total. This gives the adjusted total.*
5. *Divide the adjusted total by 1 less than the number of test scores. This is the average.*
6. *Display the average.*

Lists
Assignments

Total Sales

Design a program that asks the user to enter a store’s sales for each day of the week. The amounts should be stored in a list. Use a loop to calculate the total sales for the week and display the result

Rainfall Statistics

Rainfall Statistics
Design a program that lets the user enter the total rainfall for each of 12 months into a list. The program should calculate and display the total rainfall for the year, the average monthly rainfall, and the months with the highest and lowest amounts.

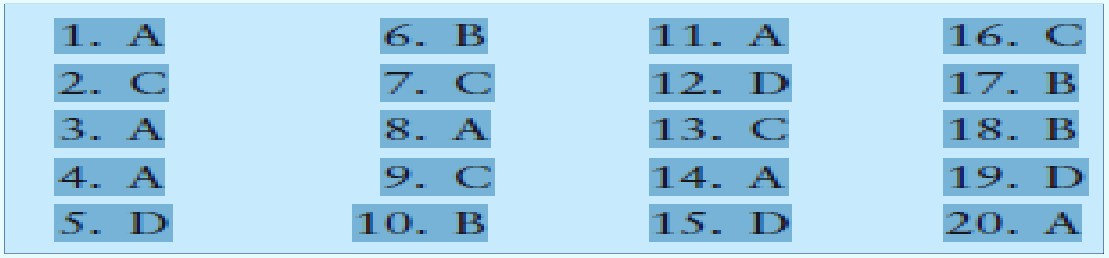
Lists

Assignments

Driver’s License Exam

Driver’s License Exam

The local driver’s license office has asked you to create an application that grades the written portion of the driver’s license exam. The exam has 20 multiple-choice questions. Here are the correct answers:



1. A	6. B	11. A	16. C
2. C	7. C	12. D	17. B
3. A	8. A	13. C	18. B
4. A	9. C	14. A	19. D
5. D	10. B	15. D	20. A

Your program should store these correct answers in a list. The program should read the student’s answers for each of the 20 questions from a text file and store the answers in another list. (Create your own text file to test the application.) After the student’s answers have been read from the file, the program should display a message indicating whether the student passed or failed the exam. (A student must correctly answer 15 of the 20 questions
to pass the exam.) It should then display the total number of correctly answered questions, the total number of incorrectly answered questions,

INTRODUCING PYTHON TUPLES

Python Tuples

A tuple is another sequence data type that is similar to the list.

A tuple consists of a number of values separated by commas.

tuples are enclosed within parentheses. (( )) and cannot be updated.

Lists are enclosed in brackets ([ ]) and their elements and size can be changed,

Tuples can be thought of as **read-only** lists. .

Once the tuple is created, you cannot add new elements, replace elements or reorder the elements in the tuple.

Tuples are more efficient than lists due Python’s implementation

```
tuple1=("Green","Red","Blue")
#Create a tuple
print(tuple1)
tuple2=tuple([7,1,2,23,4,5])  #Create a
tuple from a list
print(tuple2)
print("length is",len(tuple2)) #use the
function len
print("Maximum is",max(tuple2)) #use
the function max
print("Minimum is",min(tuple2)) #use
the function min
print("Sum is",sum(tuple2)) #use the
function sum
print("First element is",tuple2[0])
tuple3=tuple1+tuple2
print("tuple3 is",tuple3)
tuple4=2*tuple3
print("tuple4 is",tuple4)
print("tuple2[2:4] is",tuple2[2:4])
print(2 in tuple2)
```

```
for v in tuple1:
    print(v, end='')
    print()

list1=list(tuple1)
list2=list(tuple2)
print("tuple1=",tuple1)
print("tuple2=",tuple2)
print("list1 = ", list1)
print("list2 = ", list2)

tuple5=tuple(list1)
tuple6=tuple(list2)
print("tuple5=",tuple5)
print("tuple6 = ",tuple6)
print(tuple5==tuple6)
```

Python Tuples

```
thistuple = ("apple", "banana",
"cherry")
print(thistuple)
```

```
#Finding Length of a Tuple
# Code for printing the length of a
tuple
tuple2 = ('python', 'geek')
print(len(tuple2))
```

```
#Converting list to a Tuple
# Code for converting a list and a string into a tuple
list1 = [0, 1, 2]
print(tuple(list1))
print(tuple('python')) # string 'python'
```

```
#Tuples in a loop
#python code for creating tuples in a loop

tup = ('geek',)
n = 5  #Number of time loop runs
for i in range(int(n)):
    tup = (tup,)
    print(tup)
```

Python Tuples

```
#Using cmp(), max() , min()
# A python program to demonstrate the use of
# cmp(), max(), min()

tuple1 = ('python', 'geek')
tuple2 = ('coder', 1)

if (cmp(tuple1, tuple2) != 0):

    # cmp() returns 0 if matched, 1 when not tuple1
    # is longer and -1 when tuple1 is shoter
    print('Not the same')
else:
    print('Same')
print ('Maximum element in tuples 1,2: ' +
        str(max(tuple1)) +  ',' +
        str(max(tuple2)))
print ('Minimum element in tuples 1,2: ' +
     str(min(tuple1)) + ','  + str(min(tuple2)))
```

Python Tuples

```
tuple  = ( 'abcd', 786 , 2.23, 'john',
70.2 )
```

```
tuple  = ( 'abcd', 786 , 2.23, 'john',
70.2 )
```

```
tinytuple  = [123, 'john']
```

```
print tuple # Prints complete list
```

```
('abcd', 786, 2.23, 'john', 70.200000000000003)
```

```
print tuple[0] # Prints first element of the list
```

```
abcd
```

```
print tuple[1:3] # Prints elements starting from 2nd till 3rd
```

```
[786, 2.23]
```

```
print tuple[2:] # Prints elements starting from 3rd element
```

```
[2.23, 'john', 70.200000000000003]
```

```
print tinytuple * 2 # Prints list two times
```

```
[123, 'john', 123, 'john']
```

```
print tuple + tinytuple # Prints concatenated lists
```

```
['abcd', 786, 2.23, 'john', 70.200000000000003, 123,
'john']
```

```
tuple[2] = 1000 # Invalid syntax with tuple
```

```
list[2] = 1000 # Valid syntax with list
```

Data Type Conversion

A tuple is another sequence data type that is similar to the list.

A tuple consists of a number of values separated by commas.

tuples are enclosed within parentheses.

Lists are enclosed in brackets ([ ]) and their elements and size can be changed,

while tuples are enclosed in parentheses (( )) and cannot be updated.

Tuples can be thought of as **read-only** lists. .

Converting Between Lists and Tuples

```
number_tuple = (1, 2, 3)
number_list = list(number_tuple)
print(number_list)
str_list = ['one', 'two', 'three']
str_tuple = tuple(str_list)
print(str_tuple)
```

INTRODUCING PYTHON DICTIONARY

Python Dictionary

A dictionary is an object that stores a collection of data. Each element in a dictionary has two parts: a key and a value. You use a key to locate a specific value.

In Python, a dictionary is an object that stores a collection of data.

Python's dictionaries are kind of hash table type.

Each element that is stored in a dictionary has two parts: a key and a value.

They work like associative arrays or hashes found in Perl and consist of key-value pairs. .

A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

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

Key-value pairs are often referred to as mappings because each key is mapped to a value.

Python Dictionary

Python dictionaries are similar to lists in that they are mutable and can be nested to any arbitrary depth (constrained only by available memory).

A dictionary can contain any type of Python object, including another dictionary.

The keys in a given dictionary do not need to be the same type as one another, nor do the values.

Dictionary elements are accessed by key. Unlike with list indexing, the order of the items in a dictionary plays no role in how the items are accessed.

Python Dictionary

Dictionaries and lists share the following characteristics:

Both are mutable.

Both are dynamic. They can grow and shrink as needed

Both can be nested. A list can contain another list.

A dictionary can contain another dictionary.

A dictionary can also contain a list, and vice versa.

Python Dictionary

Dictionaries differ from lists primarily in how elements are accessed:

List elements are accessed by their position in the list, via indexing.

Dictionary elements are accessed via keys.

Python Dictionary

Dictionary stores (key, value) pairs, similar to a Map in Java or an object in Javascript. You can use it like this:

```
d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with
some data

print(d['cat'])       # Get an entry from a dictionary; prints "cute"

print('cat' in d)     # Check if a dictionary has a given key; prints
"True"

d['fish'] = 'wet'     # Set an entry in a dictionary

print(d['fish'])      # Prints "wet"

# print(d['monkey'])  # KeyError: 'monkey' not a key of d

print(d.get('monkey', 'N/A'))  # Get an element with a default;
prints "N/A"

print(d.get('fish', 'N/A'))    # Get an element with a default; prints
"wet"
```

Python Dictionary

Loops: It is easy to iterate over the keys in a dictionary:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

Python Dictionary

If you want access to keys and their corresponding values, use the items method:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A
spider has 8 legs"
```

Python Dictionary

Dictionary comprehensions: These are similar to list comprehensions, but allow you to easily construct dictionaries. For example:

```
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2
== 0}
print(even_num_to_square)  # Prints "{0: 0, 2: 4, 4: 16}"
```

Python Dictionary

Create a dictionary by enclosing the elements inside a set of curly braces ({}).

```
phonebook = {'Chris':'555−1111', 'Katie':'555−2222', 'Joanne':'555-3333'}
```

Remember that string comparisons are case sensitive.

The expression phonebook['katie'] will not locate the key 'Katie' in the dictionary.

Python Dictionary

```
dict = {}
```

```
dict['one'] = "This is one"
```

```
dict[2] = "This is two"
```

```
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
```

```
print dict['one'] # Prints value for 'one' key
```

```
This is one
```

```
print dict[2] # Prints value for 2 key
```

```
This is two
```

```
print tinydict # Prints complete dictionary
```

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

```
print tinydict.keys() # Prints all the keys
```

```
['dept', 'code', 'name']
```

```
print tinydict.values() # Prints all the values
```

```
['sales', 6734, 'john']
```

Dictionaries have no concept of order among elements

It is incorrect to say that the elements are "out of order"; they are simply unordered.

Python Dictionary

A few of the dictionary methods

Metho d	Descripti on
clear	Clears the contents of a dictionary.
get	Gets the value associated with a specified key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.
items	Returns all the keys in a dictionary and their associated values as a sequence of tuples.
keys	Returns all the keys in a dictionary as a sequence of tuples.
pop	Returns the value associated with a specified key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.
popitem	Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary..
values	Returns all the values in the dictionary as a sequence of tuples.

Python Dictionary

The clear Method

```
phonebook = {'Chris':'555−1111', 'Katie':'555−2222'}
print(phonebook)
#The clear method deletes all the elements in a dictionary, leaving the dictionary empty.
#The method’s general format is
# dictionary.clear()
print(phonebook)
phonebook.clear()
phonebook
```

Python Dictionary

### The get Method

```
#The get Method
phonebook = {'Chris':'555−1111', 'Katie':'555−2222'}
value = phonebook.get('Katie', 'Entry not found')
print(value)
```

```
#The get Method
#get method as an alternative to the [] operator for getting a value from a dictionary.
#The get method does not raise an exception if the specified key is not found.
# Here is the method’s general format:
#dictionary.get(key, default)
phonebook = {'Chris':'555−1111', 'Katie':'555−2222'}
value = phonebook.get('Andy', 'Entry not found')
print(value)
```

Python Dictionary

### The get Method

```
dictionary.get(key, default)
```

```
#The get Method
phonebook = {'Chris':'555−1111', 'Katie':'555−2222'}
value = phonebook.get('Katie', 'Entry not found')
print(value)
```

```
#The get Method
#get method as an alternative to the [] operator for getting a value from a dictionary.
#The get method does not raise an exception if the specified key is not found.
# Here is the method’s general format:
#dictionary.get(key, default)
phonebook = {'Chris':'555−1111', 'Katie':'555−2222'}
value = phonebook.get('Andy', 'Entry not found')
print(value)
```

Python Dictionary

The items Method

The items method returns all of a dictionary’s keys and their associated values.

They are returned as a special type of sequence known as a *dictionary view*.

Each element in the dictionary view is a tuple, and each tuple contains a key and its associated value.

```
#The items Method
phonebook = {'Chris':'555−1111', 'Katie':'555−2222','Joanne':'555−3333'}
for key, value in phonebook.items():
    print(key, value)
```

Python Dictionary

The keys Method

The keys method returns all of a dictionary’s keys as a dictionary view, which is a type of sequence

Each element in the dictionary view is a key from the dictionary

```
#The keys Method
phonebook = {'Chris':'555−1111','Katie':'555−2222','Joanne':'555−3333'}
for key in phonebook.keys():
    print(key)
```

Python Dictionary

The pop Method

The pop method returns the value associated with a specified key and removes that key value pair from the dictionary.

If the key is not found, the method returns a default value

dictionary.pop(*key, default*)

```
#The pop Method
phonebook = {'Chris':'555−1111', 'Katie':'555−2222','Joanne':'555−3333'}
phone_num = phonebook.pop('Chris', 'Entry not found')
phone_num
```

Python Dictionary

The pop Method

The pop method returns the value associated with a specified key and removes that key value pair from the dictionary.

If the key is not found, the method returns a default value

dictionary.pop(*key, default*)

```
#The pop Method
phonebook = {'Chris':'555−1111', 'Katie':'555−2222','Joanne':'555−3333'}
phone_num = phonebook.pop('Chris', 'Entry not found')
phone_num
```

Python Dictionary

## How to change or add elements in a dictionary?

Dictionary are mutable.

We can add new items or change the value of existing items using assignment operator.

If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

```
#How to change or add elements in a dictionary?

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)
```

Python Dictionary

How to delete or remove elements from a dictionary?

We can remove a particular item in a dictionary by using the method pop().

This method removes as item with the provided key and returns the value.

The method, popitem() can be used to remove and return an arbitrary item (key, value) form the dictionary.

All the items can be removed at once using the clear() method.

All the items can be removed at once using the clear() method.

Python Dictionary

## How to delete or remove elements from a dictionary?

```
#How to delete or remove elements from a dictionary?

# create a dictionary
squares = {1:1, 2:4, 3:9, 4:16, 5:25}
# remove a particular item
# Output: 16
print(squares.pop(4))
# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(squares)
# remove an arbitrary item
# Output: (1, 1)
print(squares.popitem())
# Output: {2: 4, 3: 9, 5: 25}
print(squares)
# delete a particular item
del squares[5]
# Output: {2: 4, 3: 9}
print(squares)
# remove all items
squares.clear()
# Output: {}
print(squares)
# delete the dictionary itself
del squares
# Throws Error
# print(squares)
```

Python Dictionary

Dictionary Membership Test

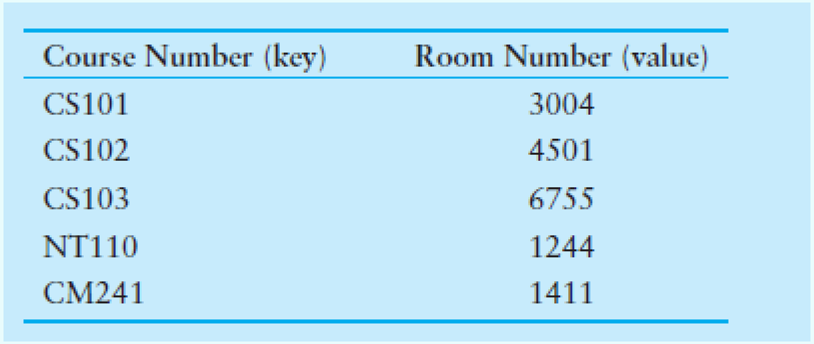
Python Dictionary

Built-in Functions with Dictionary

Function	Description
all()	Return True if all keys of the dictionary are true (or if the dictionary is empty).
any()	Return True if any key of the dictionary is true. If the dictionary is empty, return False
len()	Return the length (the number of items) in the dictionary.
cmp()	Compares items of two dictionaries
sorted()	Return a new sorted list of keys in the dictionary.

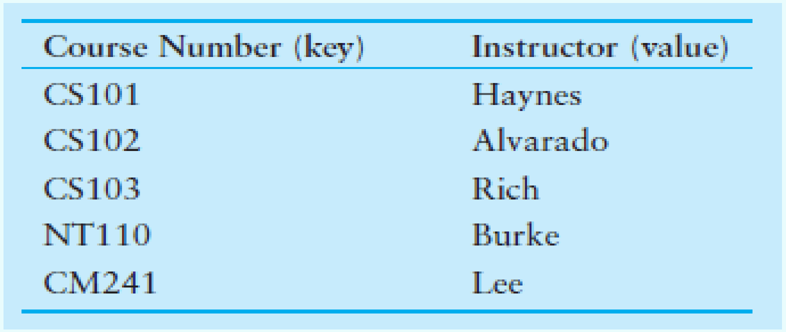
DICTIONARY
Assignments

Course information:
Write a program that creates a dictionary containing course numbers and the room numbers of the rooms where the courses meet. The dictionary should have the following key value pairs:



Course Number (key)	Room Number (value)
CS101	3004
CS102	4501
CS103	6755
NT110	1244
CM241	1411

The program should also create a dictionary containing course numbers and the names of the instructors that teach each course. The dictionary should have the following key-value pairs:



Course Number (key)	Instructor (value)
CS101	Haynes
CS102	Alvarado
CS103	Rich
NT110	Burke
CM241	Lee

INTRODUCING PYTHON SETS

Python Sets

A set contains a collection of unique values and works like a mathematical set.

A *set* is an object that stores a collection of data in the same way as mathematical sets. Here are some important things to know about sets:

All the elements in a set must be unique. No two elements can have the same value.

Sets are unordered, which means that the elements in a set are not stored in any particular order.

The elements that are stored in a set can be of different data types.

Python Sets

A **set** contains an unordered collection of unique and immutable objects

The **set** data type is, as the name implies, a **Python** implementation of the **sets** as they are known from mathematics

Python Sets

Advantages of Python Sets

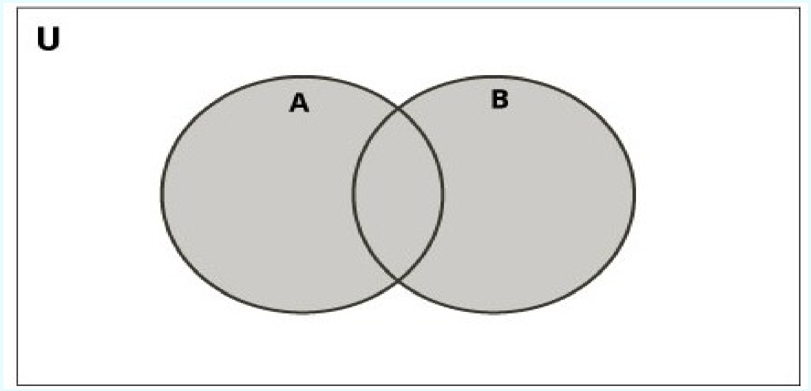
Because sets cannot have multiple occurrences of the same element, it makes sets highly useful to efficiently remove duplicate values from a list or tuple and to perform common math operations like unions and intersections.

Python Sets

Union is performed using pipe operator |operator

Compute the union of two or more sets using the 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)
```

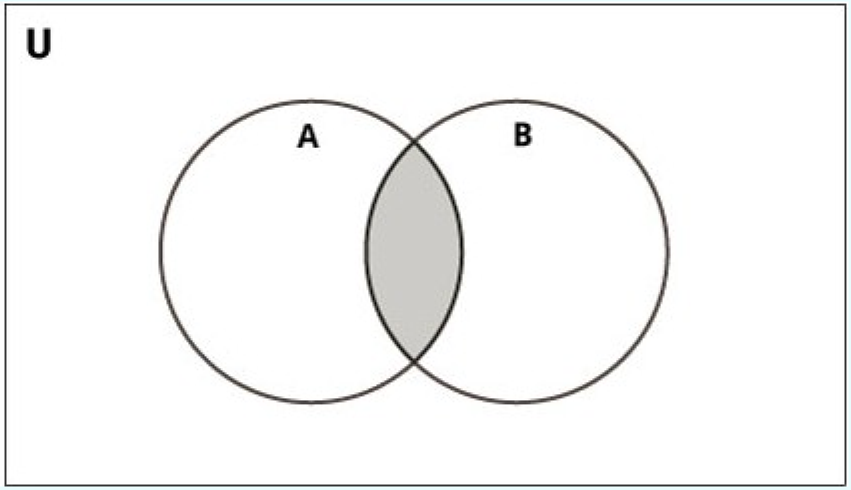


```
a = set([1, 2, 3])
b = set([3, 4, 5, 6])
c = set(list('abcd'))
print(a.union(b, c))
print(a | b | c)
# prints
set(['a', 1, 2, 3, 4, 5, 6, 'b', 'c', 'd'])
set(['a', 1, 2, 3, 4, 5, 6, 'b', 'c', 'd'])
```

Python Set Intersection

Use the intersection() method or the & operator.

Compute the union of two or more sets using the union() method.



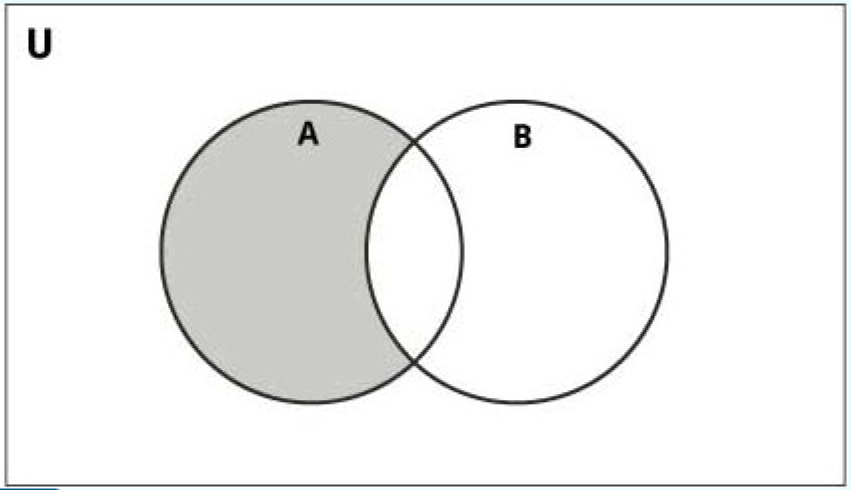
```
#Intersection
a = set([1, 2, 3])
b = set([3, 4, 5, 6])
c = set(list('abcd'))
print(a & b)
print(a & b & c)
# prints
set([3])
set([])
```

Python Set Difference

Difference of A and B (A - B) is a set of elements that are only in A but not in B

Similarly, B - A is a set of element in B but not in A.

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



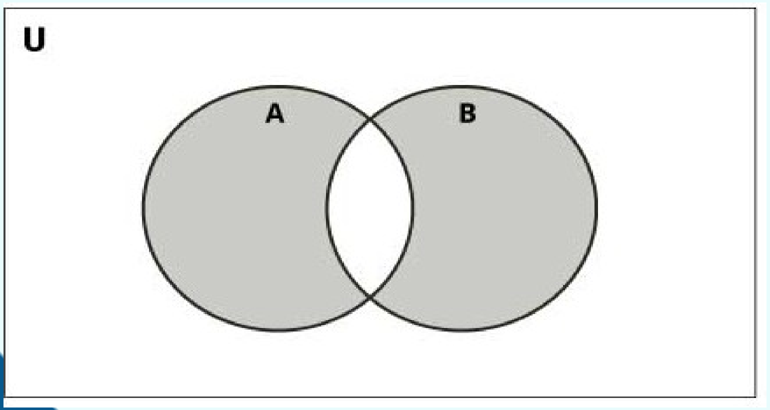
```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

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

# Python Set Symmetric Difference

Symmetric Difference of A and B is a set of elements in both A and B except those that are common in both.

Symmetric difference is performed using ^ operator.

Same can be accomplished using the method symmetric_difference().



```
#Python Set Symmetric Difference
# 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)
```

Python Sets

A set is an unordered collection of distinct elements. As a simple example, consider the following

```
animals = {'cat', 'dog'} print('cat' in animals) # Check if an element is in a set; prints "True“
 print('fish' in animals) # prints "False" animals.add('fish') # Add an element to a set

print('fish' in animals) # Prints "True" print(len(animals)) # Number of elements in a set;

prints "3" animals.add('cat') # Adding an element that is already in the set does nothing

print(len(animals)) # Prints "3" animals.remove('cat') # Remove an element from a set

print(len(animals)) # Prints "2"
```

Python Sets

Loops: Iterating over a set has the same syntax as iterating over a list; however since sets are unordered, you cannot make assumptions about the order in which you visit the elements of the set:

```
animals = {'cat', 'dog', 'fish'}
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: fish", "#2: dog", "#3: cat"
```

Python Sets

Set comprehensions: Like lists and dictionaries, we can easily construct sets using set comprehensions:

```
from math import sqrt
nums = {int(sqrt(x)) for x in range(30)}
print(nums)  # Prints "{0, 1, 2, 3, 4, 5}"
```