Experiment No-01: Python Variables, Expressions and Statements.

Objectives

- Get familiar with the variables, expressions, and statements.
- Learn different operators and their various operations.

Example 1: Values and Types.

```python
print(4)
#If you are not sure what type a value has.
type('Hello, World!')
type(17)
type(3.2)
```

Example 2: Variables

```python
number = 10
number = 1.1
##
message = 'And now for something completely different'
n = 17
pi = 3.1415926535897931
# This example makes three assignments. The first assigns a
   string to a new variable
# named message; the second assigns the integer 17 to n; the
   third assigns the
# (approximate) value of to pi.
#To display the value of a variable, you can use a print
   statement:
print(n)

print(pi)
```

Examples

```python
#Example 1: Declaring and assigning value to a variable

website = "apple.com"
print(website)


# assigning a new value to website
website = "programiz.com"

print(website)

#Example 3: Assigning multiple values to multiple variables
```

```
a, b, c = 5, 3.2, "Hello"

print (a)
print (b)
print (c)

# If we want to assign the same value to multiple variables at
   once, we can do this as:

x = y = z = "same"

print (x)
print (y)
print (z)
```

Example 3: Keywords

```
# If you give a variable an illegal name, you get a syntax error:

# Illegal variable names

76trombones = 'big parade'
more@ = 1000000
class = 'Advanced Theoretical Zymurgy'

# It turns out that class is one of Pythons keywords. The
   interpreter uses keywords
# to recognize the structure of the program, and they cannot be
   used as variable
# names.
```

Example 4: Implicit Type Conversion

```
# Example 1: Converting integer to float

num_int = 123
num_flo = 1.23

num_new = num_int + num_flo

print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))

print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))

# Example 2: Addition of string(higher) data type and
```

```
   integer(lower) datatype

num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str:",type(num_str))

print(num_int+num_str)
```

Example 5: Explicit Type Conversion

```
#Example 3: Addition of string and integer using explicit
   conversion

num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))

num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))

num_sum = num_int + num_str

print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

Example 6: Operators

```
#  Example 1: Arithmetic operators in Python
x = 15
y = 4

# Output: x + y = 19
print('x + y =',x+y)

# Output: x  y = 11
print('x  y =',xy)

# Output: x * y = 60
print('x * y =',x*y)

# Output: x / y = 3.75
print('x / y =',x/y)

# Output: x // y = 3
```

```
print('x // y =',x//y)

# Output: x ** y = 50625
print('x ** y =',x**y)
```

Example 7: String Operations

```
first = 10
second = 15
print(first+second)

############
first = '100'
second = '150'
print(first + second)

#The * operator also works with strings by multiplying the
   content of a string by an integer. For example:

first = 'Test '
second = 3
print(first * second)
```

Example 8: Asking the user for input

```
inp = input("Enter a value ")
print(inp)
```

Example 9: Output formatting

```
# one way
x = 5
y = 10
print('The value of x is {} and y is {}'.format(x,y))

#Here, the curly braces {} are used as placeholders.

# second way
x = 5
y = 10
print(f'The value of x is {x} and y is {y}')

# We can even use keyword arguments to format the string.

print('Hello {name}, {greeting}'.format(greeting =
   'Goodmorning', name = 'John'))

# test
a = 5; b=6; c= 10
```

```
print(f"The summation of {a},{b}, and {c} =",a+b+c)

# We can also format strings like the old sprintf() style used
   in C programming language.
# We use the % operator to accomplish this
x = 12.3456789
print('The value of x is %.2f' %x)
print('The value of x is %.4f' %x)


# round function
round(x,4)
```

Example 10: Package

```
# Python math package
from math import sqrt,pi
sqrt(16)
```

Practice Exercise

1. Write a program that uses input to prompt a user for their name and then welcomes them.

   Sample Input-Output:

   Enter your name: Shakib
   Hello Shakib

2. Write a program to prompt the user for hours and rate per hour to compute gross pay.

   Sample Input-Output:

   Enter Hours: 35
   Enter Rate: 2.75
   Pay: 96.25

3. Write a program that prompts the user for a Celsius temperature, convert the temperature to Fahrenheit, and print out the converted temperature.

   Sample Input-Output:

Enter Temperature in Celsius: 35
Temparature in Farenheit: ???

4. Write a program that prompts the user for the height and width of a rectangle and calculate the area, diagonal, and perimeter.

   Sample Input-Output:

   Enter Height: 5
   Enter Width: 7
   Area: ??
   Diagonal: ??
   Perimeter: ??

5. Write a Python program to calculate the area of a trapezoid.

   Sample Input-Output:

   Enter Height: 5
   Enter base 1 value: 18
   Enter base 2 value: 7
   Area: ??

6. Write a Python program to calculate the surface volume and area of a cylinder.

   Sample Input-Output:

   Enter Height: 5
   Enter radius: 10
   Volume: ??
   Area : ??

7. Write a Python program to calculate the arc length of an angle and the sector area of a circle.

   Sample Input-Output:

   Enter Radius: 5
   Enter Angle: 60
   Arc Length: ??
   Sector Area: ??

Experiment No-02: Python Conditional Statements

Objectives

- Get familiar with the Python conditional statements and different conditional operators.
- Write Programs using Python if else statements.

Example 1: Boolean expressions.

```
# A boolean expression is an expression that is either true or
   false.

print(5 == 5)
#True
print(5 == 6)
#False

#True and False are special values that belong to the class
   bool; they are not strings:
print(type(True))
# <class 'bool'>
print(type(False))
#<class 'bool'>
```

Example 2: Comparison operators

```
# Comparison operators in Python
x = 10
y = 12

# Output: x > y is False
print('x > y is',x>y)

# Output: x < y is True
print('x < y is',x<y)

# Output: x == y is False
print('x == y is',x==y)

# Output: x != y is True
print('x != y is',x!=y)

# Output: x >= y is False
print('x >= y is',x>=y)

# Output: x <= y is True
```

```
print('x <= y is',x<=y)

# x is the same as y ##  x==y
print(x is y)
# x is not the same as y x!=y
print(x is not y)
```

Example 3: Logical operators

```
# Logical Operators in Python

x = True
y = False

print('x and y is',x and y)

print('x or y is',x or y)

print('not x is',not x)
```

Example 4: Identity operators

```
# Identity operators in Python

x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'

# Output: False
print(x1 is not y1)

# Output: True
print(x2 is y2)
```

Example 5: Membership operators

```
# Membership operators in Python

x = 'Hello world'

# Output: True
print('H' in x)

# Output: True
print('hello' not in x)
```

Example 6: Conditional execution

```
if x%2 == 0:
  print('x is even')
else :
  print('x is odd')
```

Example 7: Chained conditionals

```
### Example 1

x = 10
y = 15

if x < y:
  print('x is less than y')
elif x > y:
  print('x is greater than y')
else:
  print('x and y are equal')

###### Example 2

choice = input("Enter your Choice: ")

if choice == 'a':
  print('Bad guess')
elif choice == 'b':
  print('Good guess')
elif choice == 'c':
  print('Close, but not correct')
```

Example 8: Nested conditionals

```
if x == y:
  print('x and y are equal')
else:
  if x < y:
    print('x is less than y')
  else:
    print('x is greater than y')
```

Practice Exercise

1. Write a program to prompt the user for hours and rate per hour to compute gross pay. However, the employees who worked above 40 hours were given 1.5 times the hourly rate for individual hours.

   Sample Input-Output:

Enter Hours: 45
Enter Rate: 10
Pay: 475.0

2. A company decided to give a bonus of 5% to an employee if his/her year of service is more than 5 years. Ask the user for their salary and year of service and print the net bonus amount.

3. Take values of the length and breadth of a rectangle from the user and check if it is square or not.

4. Take two int values from the user and print the greatest among them.

5. A shop will give a discount of 10% if the cost of the purchased quantity is more than 1000. Ask the user for quantity Suppose, one unit will cost 100. Judge and print the total cost for the user.

6. A school has the following rules for the grading system:

   a. Below 25 - F
   b. 25 to 45 - E
   c. 45 to 50 - D
   d. 50 to 60 - C
   e. 60 to 80 - B
   f. Above 80 - A

   Ask the user to enter marks and print the corresponding grade.

7. Take input of age of 3 people by the user and determine oldest and youngest among them.

8. A student will not be allowed to sit in an exam if his/her attendance is less than 75%. Take the following input from the user:

   Number of classes held
   The number of classes attended

   And print the percentage of classes attended. Is the student allowed to sit in an exam or not?

9. Modify the above question to allow a student to sit if he/she has a medical cause. Ask the user if he/she has a medical cause or not ('Y' or 'N') and print accordingly.

10. Sort three numbers in ascending and descending order using conditional statements.

Experiment No-03: Python Iterators or Loops

Objectives

- Get familiar with the Python control flow statements.
- Write Programs using Python loops.

Example 1: For Loop.

```
# Syntax of for Loop

'''
for val in sequence:
    loop body
'''

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

#for (int i =0; i<5; i++)

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

Example 2: Range Function

```
# We can generate a sequence of numbers using range() function.
   range(10) will generate numbers from 0 to 9 (10 numbers).

# We can also define the start, stop and step size as
   range(start, stop,step_size). step_size defaults to 1 if not
   provided.

print(range(10))

print(list(range(10)))

print(list(range(2, 8)))

print(list(range(2, 20, 3)))


#function(argument)

# argument
# return value

b = "Hello World"
```

```
len(b)

numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
max(numbers)
```

Example 3:

```
# Program to find the sum of all numbers stored in a list

# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0

# iterate over the list
for number in numbers:
    sum = sum+number
print("The sum is", sum)

# Another way
sum = 0
for i in range(len(numbers)):
    sum = sum + numbers[i]
    print(sum)


# Program to wish friends with new year greetings

friends = ['Joseph', 'Glenn', 'Sally']

for friend in friends:
    print('Happy New Year:', friend)

print('Done!')

# Another way

for i in range(len(friends)):
    print('Happy New Year:', friends[i])
```

Example 4:

```
# program to count the number of items in a list
```

```python
count = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    count = count + 1
print('Count: ', count)

# computes the total of a set of numbers
total = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    total = total + itervar
print('Total: ', total)
```

Example 5:

```python
#To find the largest value in a list or sequence,

largest = None
print('Before:', largest)

for itervar in [3, 41, 12, 9, 74, 15]:
    if largest is None or itervar > largest :
        largest = itervar
    print('Loop:', itervar, largest)

print('Largest:', largest)

# Another way

numbers = [3, 41, 12, 9, 74, 15]
largest= numbers[0]

for itervar in numbers:
    if itervar > largest :
        largest = itervar
    print('Loop:', itervar, largest)

print('Largest:', largest)
```

Example 6: While Loop

```python
# Program to add natural
# numbers up to
# sum = 1+2+3+...+n

# To take input from the user,
# n = int(input("Enter n: "))

n = 10
```

```
# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1   # update counter

# print the sum
print("The sum is", sum)

### Example of infinite loop

while True:
    if inp = 'Done':
        break

# Example 3
# Print i as long as i is less than 6:
i = 1
while i < 6:
    print(i)
    i += 1
```

Example 7: Break and Continue

```
# Use of break statement inside the loop

for val in "string":
    if val == "i":
        break
    print(val)

print("The end")


# Program to show the use of continue statement inside loops

for val in "string":
    if val == "i":
        continue
    print(val)

print("The end")
```

Example 8: Exception Handling

```
try:
    # code that may cause exception
except:
    # code to run when exception occurs

# Example 1

try:
    numerator = 10
    denominator = 0

    result = numerator/denominator

    print(result)
except:
    print("Error: Denominator cannot be 0.")

# Output: Error: Denominator cannot be 0.
```

Practice Exercise

1. Write a program that repeatedly reads numbers until the user enters "done". Once "done" is entered, print out the total, count, and average of the numbers. If the user enters anything other than a number, detect their mistake using try and except and print an error message and skip to the next number.

   Sample Input-Output:

   ```
   Enter a number: 4
   Enter a number: 5
   Enter a number: bad data
   Invalid input
   Enter a number: 7
   Enter a number: done
   16 3 5.333333333333333
   ```

2. Write a program to prompt for a score between 0.0 and 1.0. If the score is out of range, print an error message. If the score is between 0.0 and 1.0, print a grade using the following table:

   Score Grade
   >= 0.9 A
   >= 0.8 B

>= 0.7 C
>= 0.6 D
< 0.6 F

Sample Input-Output:

```
Enter score: 0.95
A
Enter score: perfect
Bad score
Enter score: 10.0
Bad score
Enter score: 0.75
C
Enter score: 0.5
F
```

3. Write a Python program to find those numbers which are divisible by 7 and multiple of 5, between 1500 and 2700 (both included).
4. Write a Python program that prints all the numbers from 0 to 6 except 3 and 6.
5. Write a Python program to get the Fibonacci series between 0 to 50
6. Write a program to display all prime numbers within a range.
7. Write a program to accept a number from a user and calculate the sum of all numbers from 1 to a given number.

Experiment No-04: Python Strings and User defined functions

Objectives

- Get familiar with various operations and methods of Python strings.
- Get familiar with user-defined functions in Python.

Example 1: Strings.

```
#A string is a sequence of characters.

fruit = 'banana'
letter = fruit[1]
print(letter)

# len is a builtin function that returns the number of
   characters in a string

fruit = 'banana'
len(fruit)

#To get the last letter of a string, you might be tempted to try
   something like this:

length = len(fruit)
last = fruit[length]
#print(last)

# To get the last character, you have to subtract 1 from the
   length
```

Example 2: Traversal through a string with a loop

```
## Example
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(letter)
    index = index + 1

# Another way to write a traversal is with a for loop:

for char in fruit:
    print(char)
```

Example 3: String slices

```
## Example 1
s = 'Monty Python'
print(s[0:5])
print(s[6:12])

## Example 2
fruit = 'banana'
print(fruit[:3])

print(fruit[3:])
```

Example 4: Strings are immutable

```
# Not Possible
greeting = 'Hello, world!'
greeting[0] = 'J'

# Correct way
# String Concatenation
greeting = 'Hello, world!'
new_greeting = 'J' + greeting[1:]
print(new_greeting)
```

Example 5: Looping and counting

```
# The following program counts the number of times the letter a
   appears in a string:

word = 'banana'
count = 0
for letter in word:
   if letter == 'a':
      count = count + 1
print(count)
```

Example 6: String methods

```
##
a = 'Hello world'
type(a)
# Show all the string methods available in python
dir(a)

# some methods
```

```
w = "Hello"

w.isalpha() # checking

w.find('l') # searching

w.count('l') # counter

### convert into upper case

word = 'banana'
new_word = word.upper()
print(new_word)

# find that searches for the position of one string within
   another
word = 'banana'
index = word.find('a')
print(index)

# Another example

word = 'banana'

print(word.find('na'))

#It can take as a second argument the index where it should
   start:

print(word.find('na', 3))

# One common task is to remove white space (spaces, tabs, or
   newlines) from the beginning and end of a string using the
   strip method:


line = '     Here we go '
line.strip()

#line.lstrip()
#line.rstrip()

# Some methods such as startswith return boolean values.
line = 'Have a nice day'
print(line.startswith('Have'))
print(line.startswith('h'))
```

```
# You will note that startswith requires case to match, so
   sometimes we take a line
# and map it all to lowercase before we do any checking using
   the lower method.

line = 'Have a nice day'
line= line.lower()
line.startswith('h')

# Another way

line.lower().startswith('h')
```

Example 7: Parsing strings

Often, we want to look into a string and find a substring. For example if we were presented a series of lines formatted as follows:

From stephen.marquard@ uct.ac.za Sat Jan 5 09:14:16 2008

and we wanted to pull out only the second half of the address (i.e., **uct.ac.za**) from each line, we can do this by using the ***find*** method and ***string slicing***.

First, we will find the position of the at-sign in the string. Then we will find the position of the first space after the at-sign. And then we will use string slicing to extract the portion of the string which we are looking for.

```
# Example of parsing

data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008',

atpos = data.find('@')
print(atpos)

sppos = data.find(' ',atpos)
print(sppos)

host = data[atpos+1:sppos]
print(host)
```

Example 8: Functions

```
# arguments with return value

def addtwo(a, b):
    added = a + b
    return added


x = addtwo(3, 5)
# print(x)

# arguments with multiple return value

def arith(a,b):
    added = a + b
    sub = a b
    m = a*b
    return added, sub, m


x,y,z = arith(3)

print(x)
print(y)
print(z)
```

Practice Exercise

1. Take the following Python code that stores a string.

   str = 'X-DSPAM-Confidence:0.8475'

   Use **find** and **string slicing** to extract the portion of the string after the colon character and then use the float function to convert the extracted string into a floating point number.

2. Write a Python program to get a string made of the first 2 and the last 2 chars from a given string. If the string length is less than 2, return instead of the empty string.

   Sample Input-Output:

   Sample String : 'w3resource'
   Expected Result : 'w3ce'
   Sample String : 'w3'
   Expected Result : 'w3w3'
   Sample String : ' w'
   Expected Result : Empty String

3. Write a Python program to add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add

'ly' instead. If the string length of the given string is less than 3, leave it unchanged. (use endswith).

Sample Input-Output:

Sample String : 'abc'
Expected Result : 'abcing'
Sample String : 'string'
Expected Result : 'stringly'

4. Write a Python program to find the first appearance of the substring 'not' and 'poor' from a given string, if 'not' follows the 'poor', replace the whole 'not'...'poor' substring with 'good'. Return the resulting string.

   Sample Input-Output:

   Sample String : 'The lyrics is not that poor!'
   'The lyrics is poor!'
   Expected Result : 'The lyrics is good!'
   'The lyrics is poor!'

5. Rewrite your pay computation (from lab-02) with time-and-a-half for overtime and create a function called compute pay which takes two parameters (hours and rate).

   Sample Input-Output:

   Enter Hours: 45
   Enter Rate: 10
   Pay: 475.0

6. Rewrite the grade program from the previous lab(03) using a function called computegrade that takes a score as its parameter and returns a grade as a string.

   Sample Input-Output:

   Enter score: 0.95
   A
   Enter score: perfect
   Bad score
   Enter score: 10.0
   Bad score
   Enter score: 0.75
   C
   Enter score: 0.5
   F

Experiment No-05: Python Files and Lists

Objectives

- Get familiar with Python Files.
- Get familiar with the Lists and its various operations .

Helper files: Strings.

```
  # download the text file using the following command
!wget https://www.py4e.com/code3/mboxshort.txt
```

Example 1: Reading Files.

```

#reading a file where each line ends with a newline character
fhand = open('mboxshort.txt')

for line in fhand:
  print(line)

#count how many lines the file have
fhand = open('mboxshort.txt')
count = 0
for line in fhand:
  count = count + 1
print('Line Count:', count)


# read method converts whole text file into a string

fhand = open('mboxshort.txt')
inp = fhand.read()
print(len(inp))
```

Example 2: Searching through a file

When you are searching through data in a file, it is a very common pattern to read through a file, ignoring most of the lines and only processing lines which meet a particular condition. We can combine the pattern for reading a file with string methods to build simple search mechanisms.

For example, if we wanted to read a file and only print out lines that started with the prefix **"From:"**, we could use the string method **startswith** to select only those lines with the desired prefix:

```

fhand = open('mboxshort.txt')
```

```
for line in fhand:
  line = line.rstrip()
  if line.startswith('From:'):
    print(line)

# We can structure the loop to follow the pattern of skipping
   uninteresting # lines as follows:


fhand = open('mboxshort.txt')
for line in fhand:
  line = line.rstrip()
  # Skip 'uninteresting lines'
  if not line.startswith('From:'):
    continue
  # Process our 'interesting' line
  print(line)
```

Example 3: Find Method

We can use the find string method to simulate a text editor search that finds lines where the search string is anywhere in the line. Since **find** looks for an occurrence of a string within another string and either returns the position of the string or -1 if the string was not found, we can write the following loop to show lines that contain the string **"@uct.ac.za"** (i.e., they come from the University of Cape Town in South Africa):

```
fhand = open('mboxshort.txt')
for line in fhand:
  line = line.rstrip()
  if line.find('@uct.ac.za') == 1:
    continue
  print(line)
```

Example 4: Traversing a list

```
cheeses = ['Cheddar', 'Edam', 'Gouda']
numbers = [17, 123]

for cheese in cheeses:
  print(cheese)

for i in range(len(numbers)):
  numbers[i] = numbers[i] * 2

print(numbers)
```

Example 5: List methods

```
# Python provides methods that operate on lists. For example,
   append adds a new
# element to the end of a list:

t = list()

t = ['a', 'b', 'c']
t.append('d')
print(t)

#extend takes a list as an argument and appends all of the
   elements:

t1 = ['a', 'b', 'c']
t2 = ['d', 'e']
t1.extend(t2)
print(t1)

#sort arranges the elements of the list from low to high
t = ['d', 'c', 'e', 'b', 'a']
t.sort()
print(t)
```

Example 6: Lists and strings

```
# A string is a sequence of characters and a list is a sequence
   of values, but a list of characters is not the same as a
   string. To convert from a string to a list of characters, you
   can use list:

s = 'spam'
t = list(s)
print(t)

# The list function breaks a string into individual letters. If
   you want to break a string into words, you can use the split
   method

s = 'pining for the fjords'
t = s.split()
print(t)
print(t[2])

# You can call split with an optional argument called a
```

```
   delimiter that specifies which characters to use as word
   boundaries. The following example uses a hyphen as a
   delimiter:

s = 'spamspamspam'
delimiter = ''
s.split(delimiter)

# join is the inverse of split. It takes a list of strings and
   concatenates the elements. join is a string method, so you
   have to invoke it on the delimiter and pass the list as a
   parameter:

t = ['pining', 'for', 'the', 'fjords']
delimiter = ' '
delimiter.join(t)
```

Example 8: Parsing lines

Usually when we are reading a file we want to do something to the lines other than just printing the whole line. Often we want to find the "interesting lines" and then parse the line to find some interesting part of the line. What if we wanted to print out the day of the week from those lines that start with "From"?

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

The split method is very effective when faced with this kind of problem. We can write a small program that looks for lines where the line starts with "From", split those lines, and then print out the third word in the line:

```
#### Append into list

week_day = list()

fhand = open('mboxshort.txt')
for line in fhand:
  line = line.rstrip()
  if line.startswith('From '):
    words = line.split()
    week_day.append(words[3])

week_day
```

Practice Exercise

1. Write a program to read through a file and print the contents of the file (line by line) all in upper case. Executing the program will look as follows: (mbox-data)

   Sample Input-Output:

   ```
   Enter a file name: mbox-short.txt
   FROM STEPHEN.MARQUARD@UCT.AC.ZA SAT JAN 5 09:14:16 2008
   RETURN-PATH: <POSTMASTER@COLLAB.SAKAIPROJECT.ORG>
   RECEIVED: FROM MURDER (MAIL.UMICH.EDU [141.211.14.90])
   BY FRANKENSTEIN.MAIL.UMICH.EDU (CYRUS V2.3.8) WITH LMTPA;
   SAT, 05 JAN 2008 09:14:16 -0500
   ```

2. Write a program to prompt for a file name, and then read through the file and look for lines of the form:

   X-DSPAM-Confidence: 0.8475

   When you encounter a line that starts with "X-DSPAM-Confidence:" pull apart the line to extract the floating-point number on the line. Count these lines and then compute the total of the spam confidence values from these lines. When you reach the end of the file, print out the average spam confidence. (mbox data)

3. Find all unique words in a file. Shakespeare used over 20,000 words in his works. But how would you determine that? How would you produce a list of all the words that Shakespeare used? Would you download all his work, read it, and track all unique words by hand? Let's use Python to achieve that instead. List all unique words, sorted in alphabetical order, that are stored in a file romeo.txt containing a subset of Shakespeare's work. To get started, download a copy of the file (https://www.py4e.com/code3/romeo.txt). Create a list of unique words, which will contain the final result. Write a program to open the file romeo.txt and read it line by line. For each line, split the line into a list of words using the split function. For each word, check to see if the word is already in the list of unique words. If the word is not in the list of unique words, add it to the list. When the program completes, sort and print the list of unique words in alphabetical order.

   Sample Input-Output:

   ```
   Enter file: romeo.txt
   'Arise', 'But', 'It', 'Juliet', 'Who', 'already',
   'and', 'breaks', 'east', 'envious', 'fair', 'grief',
   ```

```
'is', 'kill', 'light', 'moon', 'pale', 'sick', 'soft',
'sun', 'the', 'through', 'what', 'window',
'with', 'yonder'
```

4. Remove the following stopwords from the romeo.txt file and then find the list of unique words.

   stp = ["But",'is','the','with','and']

   Sample Input-Output:

   ```
   soft what light through yonder window breaks
   It east Juliet sun
   Arise fair sun kill envious moon
   Who already sick pale grief
   ```

5. Minimalist Email Client. MBOX (mailbox) is a popular file format to store and share a collection of emails. This was used by early email servers and desktop apps. Without getting into too many details, MBOX is a text file, which stores emails consecutively. Emails are separated by a special line that starts with From (notice the space). Importantly, lines starting with From: (notice the colon) describes the email itself and do not act as a separator. Imagine you wrote a minimalist email app, that lists the email of the senders in the user's Inbox and counts the number of emails. Write a program to read through the mailbox data and when you find the line that starts with "From", you will split the line into words using the split function. We are interested in who sent the message, which is the second word on the From line. (mbox-data).

   From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

   You will parse the From line and print out the second word for each From line, then you will also count the number of From (not From:) lines and print out a count at the end. This is a good sample output with a few lines removed:

   Sample Input-Output:

   ```
   Enter a file name: mbox-short.txt
   stephen.marquard@uct.ac.za
   louismedia.berkeley.edu
   zqianumich.edu
   ...some output removed...
   raymedia.berkeley.edu
   cweniupui.edu
   cweniupui.edu
   cweniupui.edu
   There were 27 lines in the file with From as the first word
   ```

6. Rewrite the program that prompts the user for a list of numbers and prints out the maximum and minimum of the numbers at the end when the user enters “done”. Write the program to store the numbers the user enters in a list and use the max() and min() functions to compute the maximum and minimum numbers after the loop completes.

   Sample Input-Output:

   ```
   Enter a number: 6
   Enter a number: 2
   Enter a number: 9
   Enter a number: 3
   Enter a number: 5
   Enter a number: done
   Maximum: 9.0
   Minimum: 2.0
   ```

Experiment No-06: Python Dictionaries and Tuples

Objectives

- Get familiar with Python Dictionaries and Tuples.
- Solving some advanced text analysis problems using them.

Helper files: Strings.

```
# download the text file using the following command
!wget https://www.py4e.com/code3/romeo.txt

!wget https://www.py4e.com/code3/romeofull.txt
```

Example 1: Dictionary.

```
eng2sp = dict()
print(eng2sp)

eng2sp['one'] = 'uno'
eng2sp['two'] = 'dos'

print(eng2sp)
#This line creates an item that maps from the key 'one' to the
   value uno.

d = {"M":"Mercuray","P":"Planet"}
print(d)

# you can create a new dictionary with three items.
eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
print(eng2sp)

# you use the keys to look up the corresponding values
print(eng2sp['two'])

# The len function works on dictionaries; it returns the number
   of keyvalue pairs:
len(eng2sp)

eng2sp['uno'] = 'uno'

print('one' in eng2sp)
print('uno' in eng2sp)

list(eng2sp.keys())
```

```
vals = list(eng2sp.values())
print('uno' in vals)
```

Example 2: Dictionary as a set of counters

```
# Suppose you are given a string and you want to count how many
   times each letter appears.

word = 'brontosaurus'
d = dict()
for c in word:
  if c not in d:
    d[c] = 1
  else:
    d[c] = d[c] + 1
print(d)


# Dictionaries have a method called get that takes a key and a
   default value. If the key appears in the dictionary, get
   returns the corresponding value; otherwise, it returns the
   default value. For example:

counts = { 'liza' : 1 , 'annie' : 42, 'jan': 100}
print(counts.get('jan', 0))
print(counts.get('tim', 0))

# We can use get to write our histogram loop more concisely.
   Because the get method automatically handles the case where a
   key is not in a dictionary, we can reduce four lines down to
   one and eliminate the if statement.

word = 'brontosaurus'
d = dict()
for c in word:
  d[c] = d.get(c,0) + 1


print(d)
```

Example 3: Dictionaries and files

One of the common uses of a dictionary is to count the occurrence of words in a file with some written text.

```
fname = input('Enter the file name: ')
try:
  fhand = open(fname)
except:
  print('File cannot be opened:', fname)
  exit()

# wordfrequency
counts = dict()
for line in fhand:
  words = line.split()
  for word in words:
    if word not in counts:
      counts[word] = 1
    else:
      counts[word] = counts[word]+ 1
print(len(counts))
```

Example 4: Looping and dictionaries

```
# If you use a dictionary as the sequence in a for statement, it
   traverses the keys of the dictionary.

counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
for key in counts:
  print(key, counts[key])

#the keys are in no particular order.

# if we wanted to find all the entries in a dictionary with a
   value above ten
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}

for key in counts:
  if counts[key] > 10 :
    print(key, counts[key])

# The for loop iterates through the keys of the dictionary,
# so we must use the index operator to retrieve the
   corresponding value for each key.

counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
lst = list(counts.keys())
print(lst)
lst.sort()
for key in lst:
  print(key, counts[key])
```

Example 5: Advanced Text Parsing

```
romeo = ["But, soft! what light through yonder window
   breaks?\nIt is the east, and Juliet is the sun.\nArise, fair
   sun, and kill the envious moon,\nWho is already sick and pale
   with grief,"]

for line in romeo:
  print(line)

#Since the Python split function looks for spaces and treats
   words as tokens separated by spaces, we would treat the words
   soft! and soft as different words and create a separate
   dictionary entry for each word. Also since the file has
   capitalization, we would treat who and Who as different words
   with different counts.

#We can solve both these problems by using the string methods
   lower, punctuation, and translate. The translate is the most
   subtle of the methods. Here is the documentation for
   translate:

import string
string.punctuation

fname = input('Enter the file name: ') ## romeofull.txt
try:
  fhand = open(fname)
except:
  print('File cannot be opened:', fname)
  exit()

counts = dict()
for line in fhand:
  line = line.rstrip()
  line = line.translate(line.maketrans('', '',
     string.punctuation)) # punctuation removal code
  line = line.lower() # make lowercase
  words = line.split()
  for word in words:
    if word not in counts:
      counts[word] = 1
    else:
      counts[word] += 1
print(counts)
```

Example 6: Tuples

```
#Dictionaries have a method called items that returns a list of
   tuples, where each tuple is a keyvalue pair:


d = {'a':10, 'b':1, 'c':22}
t = list(d.items())
print(t)

for key,val in d.items():
  print(key,val)


#As you should expect from a dictionary, the items are in no
   particular order.

#However, since the list of tuples is a list, and tuples are
   comparable, we can now sort the list of tuples. Converting a
   dictionary to a list of tuples is a way for us to output the
   contents of a dictionary sorted by key:

d = {'b':1, 'a':10,'c':22}
t = list(d.items()) ## return list of tuples
print(t)
t.sort()
print(t)

#The new list is sorted in ascending alphabetical order by the
   key value.
```

Example 7: Multiple assignment with dictionaries

```
#Combining items, tuple assignment, and for, you can see a nice
   code pattern for traversing the keys and *values *of a
   dictionary in a single loop

for key, val in list(d.items()):
  print(val, key)

#If we combine these two techniques, we can print out the
   contents of a dictionary sorted by the value stored in each
   keyvalue pair

d = {'a':10, 'b':1, 'c':22}
l = list()
for key, val in d.items() :
  l.append( (val, key) )
```

```
print(l)

l.sort(reverse=True)
print(l)
```

Example 8: The most Common Words

```
import string
fhand = open('romeofull.txt')
counts = dict()

for line in fhand:
  line = line.translate(str.maketrans('', '',
     string.punctuation))
  line = line.lower()
  words = line.split()
  for word in words:
    if word not in counts:
      counts[word] = 1
    else:
      counts[word] += 1

# Sort the dictionary by value
lst = list()
for key, val in list(counts.items()):
  lst.append((val, key))

lst.sort(reverse=True)
lst
for key, val in lst[:10]:
   print(val, key)
```

Practice Exercise

1. Write a program that categorizes each mail message by which day of the week the commit was done. To do this look for lines that start with "From", then look for the third word and keep a running count of each of the days of the week. At the end of the program print out the contents of your dictionary (order does not matter).

   Sample Input-Output:

   ```
   Enter a file name: mbox-short.txt
   'Fri': 20, 'Thu': 6, 'Sat': 1
   ```

2. Write a program to read through a mail log, build a histogram using a dictionary to count how many messages have come from each email address and print the dictionary.

   Sample Input-Output:

   ```
   Enter file name: mbox-short.txt
   'gopal.ramasammycook@gmail.com': 1, 'louis@media.berkeley.edu': 3,
   'cwen@iupui.edu': 5, 'antranig@caret.cam.ac.uk': 1,
   'rjlowe@iupui.edu': 2, 'gsilver@umich.edu': 3,
   'david.horwitz@uct.ac.za': 4, 'wagnermr@iupui.edu': 1,
   'zqian@umich.edu': 4, 'stephen.marquard@uct.ac.za': 2,
   'ray@media.berkeley.edu': 1
   ```

3. Add code to the above program to figure out who has the most messages in the file. After all the data has been read and the dictionary has been created, look through the dictionary using a maximum loop to find who has the most messages and print how many messages the person has.

   Sample Input-Output:

   ```
   Enter a file name: mbox-short.txt
   cwen@iupui.edu 5
   Enter a file name: mbox.txt
   zqian@umich.edu 195
   ```

4. This program records the domain name (instead of the address) where the message was sent from instead of who the mail came from (i.e., the whole email address). At the end of the program, print out the contents of your dictionary.

   Sample Input-Output:

   ```
   Enter a file name: mbox-short.txt
   'media.berkeley.edu': 4, 'uct.ac.za': 6, 'umich.edu': 7,
   'gmail.com': 1, 'caret.cam.ac.uk': 1, 'iupui.edu': 8
   ```

5. Revise a previous program as follows: Read and parse the "From" lines and pull out the addresses from the line. Count the number of messages from each person using a dictionary. After all the data has been read, print the person with the most commits by creating a list of (count, email) tuples from the dictionary. Then sort the list in reverse order and print out the person who has the most commits.

   Sample Input-Output:

   ```
   Enter a file name: mbox-short.txt
   cwen@iupui.edu 5
   Enter a file name: mbox.txt
   ```

```
zqian@umich.edu 195
```

6. This program counts the distribution of the hour of the day for each of the messages. You can pull the hour from the "From" line by finding the time string and then splitting that string into parts using the colon character. Once you have accumulated the counts for each hour, print out the counts, one per line, sorted by hour as shown below

   Sample Input-Output:

```
Enter a file name: mbox-short.txt
04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1
```