



**Application Development using  
Python(18CS55)**

**VTU Question Paper - Solutions**

<b>MODULE 1</b>		
<b>1 a.</b>	<b>Demonstrate with example print(), input() and string replication.</b>	<b>6M</b>
Ans	<pre># A Hello world program print("Hello World") print("What is your name?") #1 my_name = input() #2 print("It is good to meet you, "+my_name) # obtain user's lucky number lucky_num=int(input("Enter your lucky number: ")) #print out 'hello' lucky_num number of times print('hello'*lucky_num) #3</pre> <p>Output: What is your name? Jenna It is good to meet you, Jenna Enter your lucky number : 3 hellohellohello</p> <p>#1 print()  <ul style="list-style-type: none"> <li>▶ print('Hello world!')</li> <li>▶ print('What is your name?') # ask for their name</li> <li>▶ print() is a <b>function</b>.</li> <li>▶ The string to be displayed is <b>passed as a value</b> to the function</li> <li>▶ Value passed to a function is called an <b>argument</b>.</li> <li>▶ The quotes just <b>begin and end the string</b> and are not printed.</li> <li>▶ It can be used to insert a blank line: print()</li> </ul> </p> <p>#2 input()  <ul style="list-style-type: none"> <li>▶ Waits for user <b>to type</b> something and hit <b>"ENTER"</b>.</li> <li>▶ myName=input()</li> <li>▶ Assigns a string to a user.</li> <li>▶ print("It is good to meet you, "+myName)</li> <li>▶ A <b>single string</b> is passed to the print() function.</li> </ul> </p> <p>#3 String Replication  <ul style="list-style-type: none"> <li>▶ String Replication operator: When used with one string and one integer values, it replicates the string.</li> <li>▶ &gt;&gt;&gt; "hello"*3</li> </ul> <pre>'alicealicealice'</pre> <ul style="list-style-type: none"> <li>▶ &gt;&gt;&gt; 4*"bob"</li> </ul> <pre>'bobbobbob'</pre> <pre>&gt;&gt;&gt; "alice"*"bob"</pre> <p>Traceback (most recent call last): File "&lt;stdin&gt;", line 1, in &lt;module&gt; TypeError: can't multiply sequence by non-int of type 'str'</p> </p>	

<b>b</b>	<b>Explain elif, for, while, break and continue statements in Python with examples of each</b>	<b>10 m</b>
Ans.	▶ There is a possibility of many possible clauses under the <b>if</b> statement.	

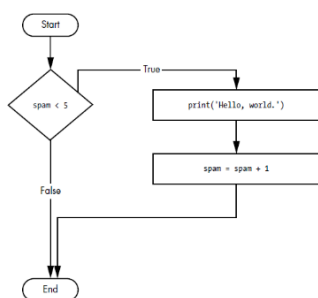
- ▶ **elif** is an **else if** that follows an if or another elif statement
- ▶ Provides another condition to be checked if previous condition evaluates to False.
  - ▶ **elif** keyword
  - ▶ **Condition** (boolean expression that evaluates to True or False)
  - ▶ Indented block of code (elif clause)
  - ▶ When there is a chain of elif clauses, **only one of them will execute.**
  - ▶ Once one of the conditions is found to be True, rest of the elif clauses are **automatically skipped.**

```

if name == "Alice":
    print("Alice")
elif age < 12:
    print("You're just a kid")
elif age < 5:
    print("you're just a baby")
  
```

### while

- ▶ Used to execute a **block of code** over and over again.



- ▶ **while** keyword
- ▶ A **condition** (expression that evaluates to True or False.)
- ▶ A **colon**
- ▶ Starting on the next line : an indented block of code – **while clause**
- ▶ In while loop, condition is **always checked** at the **beginning** of the while loop.
- ▶ The first time the condition evaluates to False **the iteration is skipped.**

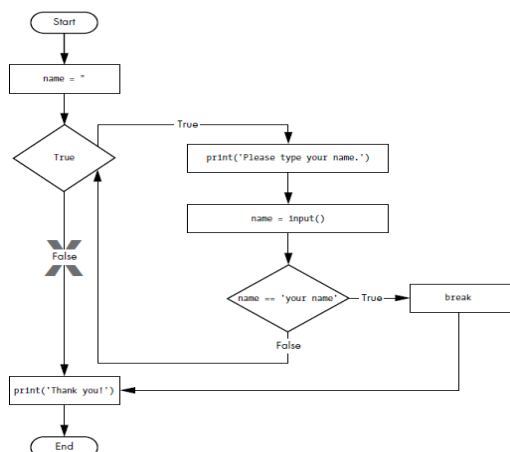
```

spam = 0
print("\nwhile statement")
while spam < 5:
    print("Hello World")
    spam = spam + 1
  
```

Output:  
while statement  
Hello World  
Hello World  
Hello World  
Hello World  
Hello World

### break

- ▶ **break** : will break out of a loop when it is encountered.



```

name = ""
while True:
    name = input("Enter your name:")
    if name == 'Max':
        break
print('Thank you')
  
```

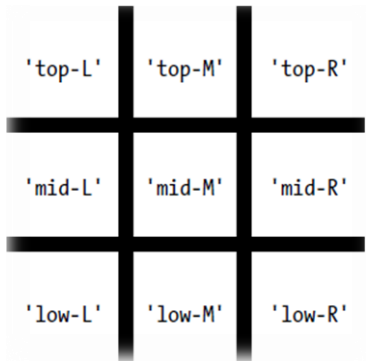
### continue

	<ul style="list-style-type: none"> <li>▶ On encountering a <b>continue statement</b>, program execution immediately jumps back to the <b>start of the loop</b>. <ul style="list-style-type: none"> <li>▶ Reevaluates the condition.</li> </ul> </li> <li>▶ This is similar to what happens when execution reaches the <b>end of the loop</b>.</li> <li>▶</li> <li>▶</li> <li>▶ <b>he loop.</b></li> </ul> <pre> while True:     print('Who are you?')     name = input()     if name != 'Max':         continue     print('Hello Max. What is your password?')     pswd = input()     if pswd == '123':         break  print('Access granted') </pre>	
<b>c.</b>	<b>Write a Python program to check whether a given number is even or odd</b>	<b>4m</b>
	<pre> num = int(input())  evenodd.py  if num % 2 == 0:     print("Number is Even") else:     print("Number is Odd")  Output: &gt; python evenodd.py 4 Number is Even  &gt; python evenodd.py 3 Number is odd </pre>	
<b>2.a.</b>	<b>How can we pass parameters in user defined functions? Explain with suitable examples.</b>	<b>5m</b>
Ans.	<ul style="list-style-type: none"> <li>▶ <b>A function is a mini-program within a program.</b></li> <li>▶ <b>Purpose:</b> group code that gets executed multiple times</li> <li>▶ Avoid duplication of code</li> <li>▶ <b>Deduplication</b> makes your programs shorter, easier to read, and easier to update</li> <li>▶ Pass values to functions (parameters)</li> <li>▶ A <b>parameter</b> is a variable that an argument is stored in when a function is called</li> <li>▶ hello("Alice") : variable name is automatically set to "Alice".</li> <li>▶ Value stored in parameter is <b>destroyed</b> when function returns</li> <li>▶ The scope is <b>local</b></li> </ul> <pre> def hello(name):     print("Hello " + name) hello("Alice") hello("Bob") print(name)  Output: </pre>	

	<p>Hello Alice Hello Bob NameError: name 'name' is not defined</p> <ul style="list-style-type: none"> <li>▶ Some arguments are identified by the position.</li> <li>▶ Eg. random.randint(1,10) signifies start, stop</li> <li>▶ <b>keyword arguments</b> are identified by the keyword put before them in the function call</li> <li>▶ print() function has two optional parameters – end and sep</li> <li>▶ end=' ' : useful in getting rid of the new line</li> </ul> <pre>&gt;&gt;&gt; print('hello', end=" ") &gt;&gt;&gt; print('cats', 'dogs', 'mice') &gt;&gt;&gt; print('cats', 'dogs', 'mice', sep=',')</pre>	
<b>b.</b>	<b>Explain local and global scope with local and global variables</b>	<b>8m</b>
Ans.	<ul style="list-style-type: none"> <li>▶ <b>Local scope:</b> Parameters and variables that are assigned in a called function <ul style="list-style-type: none"> <li>▶ Local variables : exists in a local scope</li> </ul> </li> <li>▶ <b>Global scope :</b> Variables that are assigned outside all functions <ul style="list-style-type: none"> <li>▶ Global variable : exists in a global scope</li> </ul> </li> <li>▶ <b>Scope :</b> container for variables <ul style="list-style-type: none"> <li>▶ When scope is destroyed, the variables are forgotten</li> <li>▶ <b>Global scope</b> is created when the program begins. <ul style="list-style-type: none"> <li>▶ When program terminates: global scope is destroyed.</li> </ul> </li> <li>▶ <b>Local scope</b> is created whenever function is called. <ul style="list-style-type: none"> <li>▶ Destroyed when function returns.</li> </ul> </li> </ul> </li> <li>▶ Code in the <b>global scope</b> cannot use any <b>local variables</b></li> <li>▶ a <b>local scope</b> can access <b>global variables</b></li> <li>▶ Code in a function's <b>local scope</b> cannot use variables in any other <b>local scope</b>.</li> <li>▶ Permitted to use same name for different variables if they are in <b>different scopes</b>.</li> <li>▶ Reason for scope <ul style="list-style-type: none"> <li>▶ <b>Easier to debug.</b></li> <li>▶ Always good practice to use local variables for long programs.</li> </ul> </li> <li>▶ <b>Local Variables Cannot Be Used in the Global Scope</b> <ul style="list-style-type: none"> <li>▶ Only global variables can be used in global scope.</li> </ul> </li> <li>▶ <b>Local Scopes Cannot Use Variables in Other Local Scopes</b> <ul style="list-style-type: none"> <li>▶ local variables in one function are completely separate from the local variables in another function</li> </ul> </li> </ul> <pre>def foo():     x=3     print(x) # Error as x is not defined in global scope</pre> <pre>def foo():     x = 3 # belongs to scope of foo def bar():     x+=2 # error : as x is not defined in bar() foo() bar()</pre> <ul style="list-style-type: none"> <li>▶ to modify a global variable from within a function</li> <li>▶ <b>4 rules to determine scope</b></li> </ul> <ol style="list-style-type: none"> <li>1. If variable is used in the <b>global scope</b></li> <li>2. If <b>global</b> statement is used.</li> <li>3. If variable is used in an assignment statement within a function, it is local.</li> <li>4. If variable is not used in an assignment statement within a function, it is global</li> </ol> <pre>def foo():     global x</pre>	

	<pre>print("Inside foo: x= ",x) #Inside foo: x=10 x=10 foo() print("Global value of x=",x) #Global value of x=10</pre>	
c.	<p><b>Demonstrate the concept of exception. Implement a code which prompts user for Celsius temperature, convert the temperature to Fahrenheit, and print out the converted temperature by handling the exception.</b></p>	7m
Ans.	<ul style="list-style-type: none"> <li>▶ If an error is encountered, the python program crashes.</li> <li>▶ Practically, it is better if errors were handled gracefully.</li> <li>▶ Detect errors, handle them, continue to run.</li> <li>▶ Eg. Divide-by-zero error</li> <li>▶ Syntactically the program is correct and hence these errors cannot be detected until runtime.</li> </ul> <p><b>temp_conv.py</b></p> <pre>try:     temp_celsius = float(input("Enter temperature in Celsius:"))     temp_fahrenheit = (temp_celsius *(9/5))+32     print("Temperature in Fahrenheit is : ", temp_fahrenheit) except:     print("Enter a valid value for temperature")</pre> <p>Output:</p> <pre>&gt; python temp_conv.py Enter temperature in Celsius: 45 Temperature in Fahrenheit is : 113  &gt; python temp_conv.py Enter temperature in Celsius: forty Enter a valid value for temperature</pre>	
<b>MODULE 2</b>		
3.a.	<p><b>What is a list? Explain append(), insert(), and remove() methods with examples.</b></p>	8m
Ans.	<ul style="list-style-type: none"> <li>▶ <b>list</b> is a value that contains multiple values in an ordered sequence</li> <li>▶ <b>list value</b> - value that can be stored in a variable or passed to a function like any other value</li> <li>▶ a list begins with an opening <b>square bracket</b> and ends with a closing square bracket, <b>[]</b></li> <li>▶ Items are separated with <b>commas</b> (comma delimited)</li> <li>▶ <b>[]</b> is an empty list – contains no values</li> </ul> <pre>&gt;&gt;&gt; [1,2,3] [1,2,3] &gt;&gt;&gt; ['python','dbms','os'] ['python','dbms','os'] &gt;&gt;&gt; ['Max',23,83.5,True] ['Max',23,83.5,True] &gt;&gt;&gt; student=['Max',23,83.5,True] &gt;&gt;&gt; student ['Max',23,83.5,True] &gt;&gt;&gt; type(student) &lt;class 'list'&gt; &gt;&gt;&gt; []</pre> <ul style="list-style-type: none"> <li>▶ Lists can also contain other <b>list values</b>.</li> <li>▶ If only one value is used for the index, <b>full list value is printed</b>.</li> <li>▶ If <b>two index values are used</b>, the second indicates the value in to access inside the list value.</li> <li>▶ <b>Negative Indices :</b> <ul style="list-style-type: none"> <li>▶ -1 : indicates last value in a list</li> <li>▶ -2 : indicates second last value and so on.</li> </ul> </li> </ul> <pre>&gt;&gt;&gt; student=['Max',23,[75,80,62]]</pre>	

	<pre>&gt;&gt;&gt; student[2] [75,80,62] &gt;&gt;&gt; marks=[52,82,92,98] &gt;&gt;&gt; marks[-1] 98     ▶ <b>append()</b> : adds argument to the end of the list.     ▶ <b>insert()</b> : can insert a value at any index in a list. ** notice that the return value of append() and insert() is <b>None</b>.     ▶ The list is modified <b>in place</b>     ▶ Methods belong to certain data types.         ▶ append() and insert() does not work with other data types – generates           an <b>Attribute Error</b> &gt;&gt;&gt; subjects = ['dbms','os','python'] &gt;&gt;&gt; subjects.append('Java') &gt;&gt;&gt; subjects.insert(1, 'aca') &gt;&gt;&gt; subjects ['dbms','aca','os','python','Java'] &gt;&gt;&gt;x=5 &gt;&gt;&gt;x.append(6) AttributeError &gt;&gt;&gt; x.insert(1,7) AttributeError      ▶ <b>remove()</b> – removes the occurrence of the value from the list         ▶ If there are multiple instances, the first instance is removed.         ▶ If it is not present, <b>ValueError</b> is generated. &gt;&gt;&gt; subjects= ['dbms', 'aca', 'os', 'python', 'Java', 'os'] &gt;&gt;&gt; subjects.remove('dbms') &gt;&gt;&gt; subjects ['aca', 'os', 'python', 'Java','os'] &gt;&gt;&gt; subjects.remove('os') &gt;&gt;&gt; subjects ['aca', 'python', 'Java','os']</pre>	
<b>b.</b>	<b>How is tuple different from a list and which function is used to convert list to a tuple.</b>	<b>5m</b>
Ans.	<ul style="list-style-type: none"> <li>▶ Tuples are typed with <b>parantheses</b>, ( and )</li> <li>▶ Tuples are <b>immutable</b></li> <li>▶ In other aspects they are identical to lists.</li> <li>▶ <b>TypeError</b> is generated if there is an attempt to modify a tuple.</li> <li>▶ A <b>singleton tuple</b> needs to be specified with <b>a value followed by a comma</b> <ul style="list-style-type: none"> <li>▶ Otherwise, Python will interpret it as a value inside a parantheses.</li> </ul> </li> <li>▶ Use tuples to convey a message that the <b>sequence</b> is not meant to change.</li> <li>▶ To use an <b>ordered sequence</b> that does not change, <b>use a tuple</b>.</li> <li>▶ Since contents of tuples do not change, Python can slightly optimize code that uses tuples as compared to lists.</li> </ul> <pre>&gt;&gt;&gt; a=tuple([1,2,3]) &gt;&gt;&gt; a (1,2,3) &gt;&gt;&gt; subjects=('dbms','os','python') &gt;&gt;&gt; subjects ('dbms','os','python') # indexes can be used to access items in tuples &gt;&gt;&gt; subjects[1] 'os' &gt;&gt;&gt; subjects[1:] # tuples can be sliced the same way as lists because                  # slicing returns a new tuple ('os','python') &gt;&gt;&gt;subjects[1]='java' # tuples are immutable, this assignment is acceptable in lists,                     # In tuples, it results in an error.</pre>	

	Type Error	
<b>c.</b>	<b>Create a function to print out a blank tic-tac-toe board</b>	<b>7m</b>
Ans.	<ul style="list-style-type: none"> <li>▶ looks like a large <b>hash symbol (#)</b></li> <li>▶ <b>nine slots</b></li> <li>▶ Each can contain an <b>X, an O, or a blank.</b></li> <li>▶ To represent using a <b>dictionary</b>, each slot can be given a <b>string-value key.</b></li> </ul>  <ul style="list-style-type: none"> <li>▶</li> </ul> <pre> tic-tac.py tictac = {'low-L': ' ', 'low-M': ' ', 'low-R': ' ',           'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',           'top-L': ' ', 'top-M': ' ', 'top-R': ' '} def print_board():     print(tictac['top-L']+' '+tictac['top-M']+' '+tictac['top-R'])     print('-+-+-')     print(tictac['mid-L']+' '+tictac['mid-M']+' '+tictac['mid-R'])     print('-+-+-')     print(tictac['low-L']+' '+tictac['low-M']+' '+tictac['low-R']) print_board() </pre> <p>Output:</p> <pre>     -+-+     -+-+     </pre>	
<b>4.a.</b>	<b>Discuss get(), items(), keys(), values() dictionary methods in Python with examples.</b>	<b>8m</b>
Ans.	<ul style="list-style-type: none"> <li>▶ Tedious to check if key exists before accessing the value.</li> <li>▶ get() : takes 2 arguments <ul style="list-style-type: none"> <li>▶ Key of the value to retrieve</li> <li>▶ Fallback value to return if the key does not exist.</li> </ul> </li> </ul> <pre> &gt;&gt;&gt; items={'pen':5, 'stapler':1, 'marker': 3} &gt;&gt;&gt; print("I have",items.get('pen',0), 'pens') I have 5 pens &gt;&gt;&gt; print("I have",items.get('pencil',0), 'pencils') I have 0 pencils </pre> <p>Keys(), values(), items()</p> <ul style="list-style-type: none"> <li>▶ Returns <b>list-like</b> values</li> <li>▶ These lists returned <b>are not true lists</b>: They <b>cannot be modified</b> and <b>do not have an append()</b> method.</li> <li>▶ Their data types are <b>dict_keys, dict_values, dict_items</b> <ul style="list-style-type: none"> <li>▶ Can be used in for loops.</li> </ul> </li> </ul> <p>Create a dictionary of items. Ask user to enter the name of item and the number of it.</p>	



	<p>If item exists, add the number to the existing count.  If item does not exist, create new item with the number</p> <pre>items={'pen':5, 'stapler':1, 'marker': 3} while True:     print("Enter item(blank to quit):")     item = input()     if item=="":         break     number = int(input("Number:"))     items[item]=items.get(item,0)+number print(items)</pre> <p>Sample output</p> <pre>#items={'pen':5, 'stapler':1, 'marker': 3} python items_get.py Enter item(blank to quit): stapler Number:3 Enter item(blank to quit): eraser Number:4 Enter item(blank to quit): {'pen': 5, 'stapler': 4, 'marker': 3, 'eraser': 4}</pre> <pre>&gt;&gt;&gt; codes = {'red': '#FF0000', 'sky-blue': '#87CEEB', 'coral': '#FF7F50'} &gt;&gt;&gt; codes.values() dict_values(['#FF0000', '#87CEEB', '#FF7F50']) &gt;&gt;&gt; codes.keys() dict_keys(['red', 'sky-blue', 'coral']) &gt;&gt;&gt; type(codes.values()) &lt;class 'dict_values'&gt; &gt;&gt;&gt; type(codes.keys()) &lt;class 'dict_keys'&gt; &gt;&gt;&gt; type(codes.items()) &lt;class 'dict_items'&gt; &gt;&gt;&gt; codes.items() dict_items([('red', '#FF0000'), ('sky-blue', '#87CEEB'), ('coral', '#FF7F50')]) &gt;&gt;&gt; list(codes.keys()) ['red', 'sky-blue', 'coral'] #use multiple assignment &gt;&gt;&gt; for color,hexa in codes.items(): ...     print('Color:',color,'Value:',hexa) Color: red Value: #FF0000 Color: sky-bule Value:#87CEEB Color: coral Value: #FF7F50</pre>	
<b>b.</b>	<b>With example code, explain join() and split() string methods</b>	<b>6m</b>
Ans.	<ul style="list-style-type: none"> <li>▶ <b>join()</b> : a list of strings that need to be joined together into a single string value</li> <li>▶ Method is <b>called on a string</b></li> <li>▶ Passed a <b>list of strings</b></li> <li>▶ <b>Returns a string</b> : concatenation of each string in the passed-in list</li> <li>▶ string join() calls on is <b>inserted between</b> each string of the list argument</li> </ul> <pre>&gt;&gt;&gt; ', '.join(['cats', 'rats', 'bats']) 'cats,rats,bats' &gt;&gt;&gt; ' '.join(['My', 'name', 'is', 'Simon']) 'My name is Simon' &gt;&gt;&gt; 'ABC'.join(['My', 'name', 'is', 'Simon']) 'MyABCnameABCisABCSimon'</pre>	

	<ul style="list-style-type: none"> <li>▶ <b>split():</b> called on a string value and returns a list of strings.</li> <li>▶ By default, <b>split occurs</b> wherever whitespace characters are seen. <ul style="list-style-type: none"> <li>▶ <b>space, tab, or newline characters</b></li> </ul> </li> <li>▶ Whitespace characters are <b>not included</b> in the strings in the returned list.</li> <li>▶ A <b>delimiter</b> may be passed to specify a different string to be split upon.</li> <li>▶ A common use of split() is to split a multiline string along the newline characters.</li> </ul> <pre>&gt;&gt;&gt; 'My name is Simon'.split() ['My', 'name', 'is', 'Simon'] &gt;&gt;&gt; 'MyABCnameABCisABCsSimon'.split('ABC') ['My', 'name', 'is', 'Simon'] &gt;&gt;&gt; 'My name is Simon'.split('m') ['My na', 'e is Si', 'on']</pre>	
<b>c.</b>	<b>Develop a program to accept a sentence from the user and display the longest word of that sentence along with its length</b>	<b>6m</b>
Ans.	<pre>Longestword.py  sentence = input() sentence=sentence.lower()  words =sentence.split() # by default it splits across whitespace.  words.sort(key=len) # to sort according to length print(words[-1], ':', len(words[-1]))  Sample output: &gt;python Longestword.py Application Development using Python development, 11  Alternatively, this can be done using a dictionary also.  sentence =input() sentence = sentence.lower() words = sentence.lower()  words=sentence.split()  wl = {}  for w in words:     wl[w] = len(w)  # we need to sort by length wl_list =[] for k,v in wl.items():     wl_list.append( (v,k))  wl_list.sort() print(wl_list[-1][1],':', wl_list[-1][0]) # prints the word and corresponding length</pre>	
<b>MODULE 3</b>		
<b>5 a.</b>	<b>What are regular expressions? Describe question mark, star, plus and dot regex symbols with suitable Python code snippet</b>	<b>9m</b>
Ans.	<p>Regular expressions are used for pattern matching. They have special characters that are interpreted for the purpose of matching patterns in text.</p> <ul style="list-style-type: none"> <li>▶ 1. Import the <b>regex module</b> with <b>import re.</b></li> <li>▶ 2. Create a <b>Regex object</b> with the <b>re.compile()</b> function. (Remember to use</li> </ul>	

a **raw string**.)

- ▶ 3. Pass the **string** you want to search into the Regex object's **search()** method.
  - ▶ This returns a **Match object**.
- ▶ 4. Call the Match object's **group()** method to return a string of the actual matched text.

### Optional matching with ?

- a **pattern** to match only optionally
- ▶ **?** character flags the group that **precedes** it as an **optional part**
- ▶ **(wo)?** : pattern **wo** is an optional group
- ▶ Match **has zero instances or one instance** text that of **wo** in it

```
>>> spider_re = re.compile(r'Spider(wo)?man')
>>> mo1 = spider_re.search('The Amazing Spiderman')
>>> mo1.group()
Spiderman
>>> mo2 = spider_re.search('The all new Spiderwoman')
>>> mo2.group()
Spiderwoman
```

Here the 'wo' is matched optionally.

### Matching 0 or more with the star

- ▶ **\*** (called the **star** or **asterisk**) means "match zero or more".
- ▶ Group that **precedes the star** can occur **any number of times** in the text
- ▶ can be **completely absent**
- ▶ Or **repeated** over and over again
- ▶ **'Spiderman' (wo)\*** part of the regex matches zero instances
- ▶ **'Spiderwoman', the (wo)\*** matches **one** instance of **wo**
- ▶ **'Spiderwowowoman', (wo)\*** matches **four instances** of **wo**

```
>>> spider_re = re.compile(r'Spider(wo)*man')
>>> mo1 = spider_re.search('The adventures of Spiderman')
>>> mo1.group()
Spiderman
>>> mo2 = spider_re.search('The adventures of Spiderwoman')
>>> mo2.group()
Spiderwoman
>>> mo3 = spider_re.search('The adventures of Spiderwowowoman')
>>> mo3.group()
Spiderwowowowoman
```

### Matching one or more with plus

- ▶ **+** (or plus) means "match one or more."
- ▶ group preceding a plus must appear at least once
- ▶ **Spider(wo)+man** 'The Amazing Spiderman' - will not return a match

```
>>> spider_re = re.compile(r'Spider(wo)+man')
>>> mo1 = spider_re.search('The Amazing Spiderman')
>>> mo1.group()

>>> mo2 = spider_re.search('The Amazing Spiderwoman')
>>> mo2.group()
Spiderwoman
```

### Wildcard character

- ▶ **.** (or **dot**) **character** : a wildcard and will match any character except **for a newline**
- ▶ **.\*** "anything."
  - ▶ **.** "any single character except the newline"
  - ▶ **\*** : "zero or more of the preceding character"

	<ul style="list-style-type: none"> <li>▶ Uses <b>greedy</b> mode</li> <li>▶ <b>Non greedy</b> mode : <code>.*?</code></li> </ul> <pre>&gt;&gt;&gt; nameRegex = re.compile(r'First Name: (.*) Last Name: (.*)') &gt;&gt;&gt; mo = nameRegex.search('First Name: Al Last Name: Sweigart') &gt;&gt;&gt; mo.group(1) 'Al' &gt;&gt;&gt; mo.group(2) 'Sweigart'</pre>	
<b>b.</b>	<b>With code snippet, explain saving variables using the shelve module and PPrint Pformat() functions.</b>	<b>6m</b>
Ans.	<ul style="list-style-type: none"> <li>▶ Used to save variables in your Python programs to binary shelf files</li> <li>▶ program can restore data to variables from the hard drive.</li> <li>▶ <b>Shelve module</b> : allows you to Save and Open features to your program.</li> <li>▶ To read and write data, call <b>shelve.open()</b> and pass file name.</li> <li>▶ Notice that 3 files, 'mydata.bak', 'mydata.dat', 'mydata.dir' get created in the current working directory. (On OS X, mydata.db is created)</li> </ul> <pre>&gt;&gt;&gt; import shelve &gt;&gt;&gt; shelfFile = shelve.open('mydata') &gt;&gt;&gt; topics=['read files','write files','append'] &gt;&gt;&gt; shelfFile['topics']=topics &gt;&gt;&gt; shelfFile.close() &gt;&gt;&gt; os.listdir() mydata.bak mydata.dat mydata.dir &gt;&gt;&gt; sf= shelve.open('mydata') &gt;&gt;&gt; list(sf.keys()) ['topics'] &gt;&gt;&gt; list(sf.values()) ['read files','write files','append']</pre> <ul style="list-style-type: none"> <li>▶ <b>pprint.pprint():</b> pretty prints contents of a dictionary.</li> <li>▶ <b>pprint.pformat():</b> return this same text as a string instead of printing it. <ul style="list-style-type: none"> <li>▶ The formatted string is easy to read</li> <li>▶ It is also syntactically correct</li> <li>▶ This string can be written to the .py file</li> <li>▶ This file can be your very own module you can import whenever you want the variable stored inside it.</li> </ul> </li> </ul> <pre>&gt;&gt;&gt; import pprint &gt;&gt;&gt; subjects = [{'code':'18CS55','name':'ADP'},{'code':'18CS51','name':'ME'}] &gt;&gt;&gt; pprint.pformat(subjects) &gt;&gt;&gt; fhand = open('mySubjects.py','w') &gt;&gt;&gt; fhand.write('subjects =' + pprint.pformat(subjects)+'\n') 80 &gt;&gt;&gt; fhand.close()  &gt;&gt;&gt; import mySubjects &gt;&gt;&gt; mySubjects.subjects [{'code': '18CS55', 'name': 'ADP'}, {'code': '18CS51', 'name': 'ME'}] &gt;&gt;&gt; mySubjects.subjects[0] {'code': '18CS55', 'name': 'ADP'}</pre>	
<b>c.</b>	<b>Write a program that reads a string with five characters which starts with 'a' and ends with 'z'. Print search successful if pattern matches string.</b>	<b>5m</b>
	<pre>Search.py  import re regex= re.compile(r'^a...z\$') str = input() if regex.search(str):     print("Search Successful")</pre>	

	<pre>else:     print("Search unsuccessful")</pre> <p>Sample output 1  &gt; python Search.py  Hello  Search unsuccessful</p> <p>Sample output 2  &gt; python Search.py  abyzz  Search Successful</p> <p>Sample output 3  &gt;python Search.py  abeizz  Search unsuccessful</p> <p>^ before 'a' matches the beginning of the string with 'a'  \$ after 'z' matches the end of the string with 'z'  That leaves us with 3 characters that should exist between 'a' and 'z' that would give us 5 characters in the string. This is represented by ....  It may also be represented as <code>.{3}</code>.</p> <p>In sample output 1, even though string is of 5 characters, it does not start nor end with a and z  In sample output 2, search is successful because the string comprises of 5 characters beginning with 'a' and ending with 'z'  In sample output 3, search is unsuccessful because even though the string starts and ends with 'a' and 'z', it consists of less than 5 characters.</p>	
--	---	--

<b>6 a.</b>	<b>Explain functions of shutil module with examples.</b>	<b>8m</b>
Ans.	<ul style="list-style-type: none"> <li>▶ Shell utilities module</li> <li>▶ Allows you to copy, move, rename, and delete files in Python programs</li> <li>▶ <b>shutil.copy(source, destination)</b> : copy the file at the <b>path source</b> to the <b>folder</b> at the <b>path destination</b> <ul style="list-style-type: none"> <li>▶ <i>Both source and destination are strings.</i></li> <li>▶ <i>If destination is a <b>filename</b>, it will be used as a <b>new name</b> for the copied file.</i></li> <li>▶ <i>If source and destination file names are the same, it results in <code>shutil.SameFileError</code>.</i></li> </ul> </li> </ul> <pre>&gt;&gt;&gt; shutil.copy('address.txt','home_address.txt') 'home_address.txt' &gt;&gt;&gt; shutil.copy('address.txt','test/address.txt') 'test/address.txt'</pre> <ul style="list-style-type: none"> <li>▶ <b>shutil.copy</b> will copy a single file</li> <li>▶ <b>copytree(source, destination)</b> will copy an entire folder and every folder inside it. <ul style="list-style-type: none"> <li>▶ Both source and destination are both strings.</li> </ul> </li> </ul> <pre>&gt;&gt;&gt; import shutil,os &gt;&gt;&gt; shutil.copytree('test', 'test2') 'test2'</pre> <ul style="list-style-type: none"> <li>▶ <b>shutil.move(source, destination)</b></li> <li>▶ return a string of the absolute path of the new location</li> <li>▶ If <b>destination</b> points to a folder, the <i>source</i> file gets <b>moved</b> into <b>destination with same file name</b></li> <li>▶ If destination is a file name then it is <b>moved and renamed</b></li> <li>▶ Folders that make up the <b>destination must exist</b></li> <li>▶ If it can't find the folder, it will assume it is a file and rename.</li> </ul>	

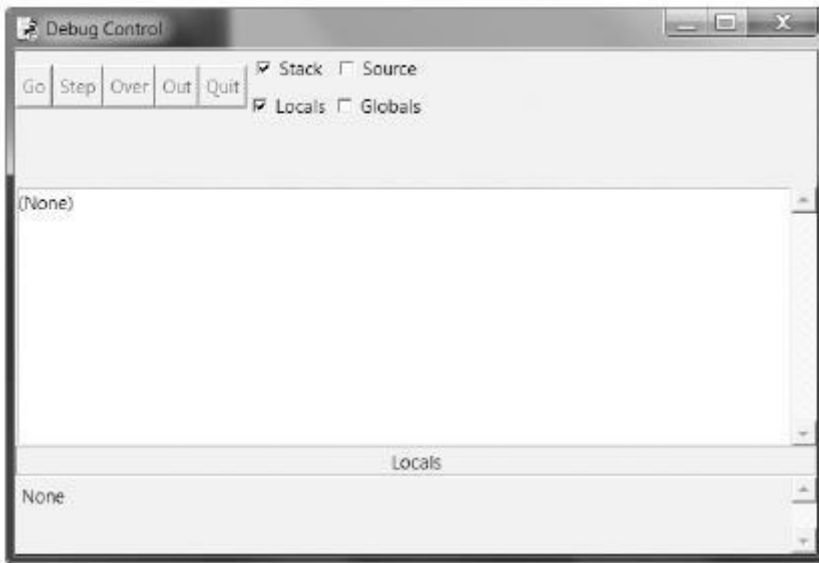
```
PS C:\Users\Admin\Documents\test> tree /f /a
Folder PATH listing
Volume serial number is 00F4-EDFD
C:.
|
| address.txt
| cron.txt
| hello.txt
|
|---foo
|
| romeo.txt
| rose.txt
```

```
>>> import shutil,os
>>> shutil.move('address.txt', 'ad.txt')
>>> shutil.move('foo', 'bar') #rename folders
>>> shutil.move('f1','f2') # rename's files
>>> shutil.move('cron.txt','C:\\Users\\Documents\\NXP')
# address is correct, but folder does not exist
>>>shutil.move('cron.txt','C:\\Users\\Documents\\NXP')
# path is incorrect
```

**b. Explain buttons in the Debug control window.**

**5m**

Ans.



**Go**

Clicking the Go button will cause the program to execute normally until it terminates or reaches a *breakpoint*.

**Step**

Clicking the Step button will cause the debugger to execute the next line of code and then pause again.

**Over**

Clicking the Over button will execute the next line of code, similar to the Step button. if the next line of code is a function call, the Over button will “step over” the code in the function

**Out**

Clicking the Out button will cause the debugger to execute lines of code at full speed until it returns from the current function.

**Quit**

If you want to stop debugging entirely and not bother to continue executing the rest of the program, click the **Quit** button.

**c. What is meant by compressing files? Explain reading, extracting, and creating ZIP files with code snippet**

**7m**

Ans.

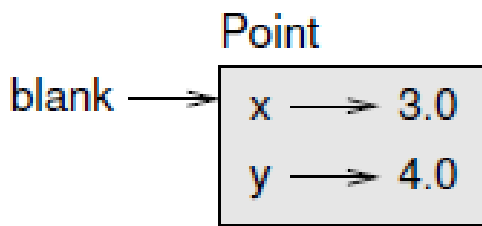
- ▶ Compressing a file reduces its size, which is useful when transferring it over the Internet.
- ▶ **namelist()** method that returns a list of strings for all the files and folders contained in the ZIP file.

	<ul style="list-style-type: none"> <li>▶ Can be passed to <b>getinfo()</b> of ZipFile <ul style="list-style-type: none"> <li>▶ To retrieve information about the particular file</li> </ul> </li> </ul> <p>Reading ZIP files</p> <pre>&gt;&gt;&gt; import zipfile, os &gt;&gt;&gt; os.chdir('C:\\Users\\Admin\\Pictures') &gt;&gt;&gt; zp = zipfile.ZipFile('sc.zip') &gt;&gt;&gt; zp.namelist() &gt;&gt;&gt; spInfo = zp.getinfo('sc/world.png') &gt;&gt;&gt; spInfo.file_size 6546 &gt;&gt;&gt; spInfo.compress_size 3096</pre> <p>ZipFile : holds information about entire object ZipInfo : holds information about particular file.</p> <p><b>Extracting from ZIP files</b></p> <ul style="list-style-type: none"> <li>▶ <b>extractall()</b> method for ZipFile objects extracts all the files and folders onto current working directory.</li> <li>▶ Pass a <b>folder name</b> to extract files into a folder other than current working directory.</li> <li>▶ If <b>folder is not found</b>, it will be <b>created</b>.</li> <li>▶ <b>extract()</b> : extracts a single file from the ZIP file.</li> <li>▶ Optional second argument : folder.</li> </ul> <pre>&gt;&gt;&gt; zp = zipfile.ZipFile('sc.zip') &gt;&gt;&gt; zp.extractall() &gt;&gt;&gt; zp.extractall('test') &gt;&gt;&gt; zp.namelist() [... ] # lists all the names of the files in a list &gt;&gt;&gt; zp.extract('sc/world.png') # extracts only world.png &gt;&gt;&gt; zp.extract('sc/world.png','test_single')</pre> <p><b>Creating and adding ZipFiles</b></p> <ul style="list-style-type: none"> <li>▶ open the <b>ZipFile</b> object in <i>write mode</i> by passing '<b>w</b>' <ul style="list-style-type: none"> <li>▶ Similar to opening file to write</li> </ul> </li> <li>▶ pass a <b>path</b> to the <b>write()</b> method of ZipFile object</li> <li>▶ 2<sup>nd</sup> argument is the <b>compression type</b> <ul style="list-style-type: none"> <li>▶ The algorithm to be used for compression</li> <li>▶ <b>zipfile.ZIP_DEFLATED</b> works well for all file types</li> </ul> </li> <li>▶ To <b>add files</b> to an <b>existing zip file</b> pass the '<b>a</b>' mode.</li> </ul> <pre>&gt;&gt;&gt; newZip = zipfile.ZipFile('sc_new.zip','w') &gt;&gt;&gt; newZip.write('download.jpg',compress_type = zipfile.ZIP_DEFLATED) &gt;&gt;&gt; newZip.write('code1.png',compress_type = zipfile.ZIP_DEFLATED) &gt;&gt;&gt; newZip.close()</pre>	
--	---	--

#### MODULE 4

<b>7 a.</b>	<b>What is class, object, attributes. Explain copy.copy() with an example.</b>	<b>6m</b>
Ans.	<ul style="list-style-type: none"> <li>▶ A <b>user-defined type</b> is also called a <b>class</b>.</li> <li>▶ Let's say you wanted to represent a point in 2D space (x,y) <ul style="list-style-type: none"> <li>▶ <b>Store it in 2 variables, x and y</b></li> <li>▶ <b>Store it as coordinates in a list or a tuple</b></li> <li>▶ <b>Create a new type to represent points as objects.</b></li> </ul> </li> <li>▶ Indicates that Point is a kind of <b>object</b> which is a built-in type.</li> <li>▶ Point is at the top level, so full name is <code>__main__.Point</code></li> <li>▶ Creating a new object is called <b>instantiation</b>.</li> <li>▶ The object is an <b>instance</b> of the class.</li> <li>▶ When you print an instance, you will get information on <ul style="list-style-type: none"> <li>▶ What <b>class</b> it belongs to</li> </ul> </li> </ul>	

► **Address** in memory



```
>>> class Point(object):
    pass
>>> type(Point)
<class 'type'>
>>> Point
<class '__main__.Point'>
```

#### Attributes / Instance variables

► Use **dot notation** to assign values to an instance.

```
>>> blank.x=3.0
>>> blank.y=4.0
```

► We are assigning **named elements** to the object.

► To get the **value**, again use dot notation

► blank.x - Go to the blank object and get the value of x.

```
>>> blank.x=3.0
>>> blank.y=4.0
>>> blank.x
3.0
>>> blank.y
4.0
```

#### Copy.copy()

Aliasing can cause problems and unexpected behaviour of code.

**copy() of the copy module** : can duplicate any object.

```
>>> class Point:
...     pass
...
>>> p=Point()
>>> p.x,p.y=3.0,4.0
>>> import copy
>>> cp = copy.copy(p)
>>> print(p.x,p.y)
3.0 4.0
>>> print(cp.x,cp.y)
3.0 4.0
>>> p is cp
False
>>> p==cp
False
```

<b>b.</b>	<b>Demonstrate pure functions and modifiers with examples</b>	<b>8m</b>
Ans.	<ul style="list-style-type: none"><li>► Pure Functions : Does not <b>modify</b> any of the objects passed to it as arguments.</li><li>► It has no effect<ul style="list-style-type: none"><li>► Getting user input</li><li>► Displaying a value</li></ul></li><li>► It only returns.</li> <li>► Functions that <b>modify</b> the objects it gets as parameters.</li><li>► Anything that can be done with <b>modifiers</b> can also be done with <b>pure</b></li></ul>	



### functions

- ▶ Some programming languages only allow **pure functions**
- ▶ Evidence suggest that only using pure functions
  - ▶ Make development faster
  - ▶ Less error prone
- ▶ **Functional programming style** – use **pure functions as building blocks** that do not have **side-effects** and **do not change the state.**
  - ▶ Resort to modifiers in case of a **compelling advantage**

Consider a function increment() that takes the time object (hour, minute, second) and seconds as the second parameter to add the seconds to the time object.

```
class Time:
```

```
    pass
```

```
# modifier
```

```
def increment(t,sec):
```

```
    t.second+=sec
```

```
    if t.second>=60:
```

```
        e_m, e_s = (t.second)//60, (t.second)%60
```

```
        t.second = e_s
```

```
        t.minute+=e_m
```

```
# pure function
```

```
def increment_pure(t,sec):
```

```
    inc_t=Time()
```

```
    inc_t.hour=t.hour
```

```
    inc_t.second=t.second+sec
```

```
    if inc_t.second>=60:
```

```
        e_m, e_s = (inc_t.second)//60, (inc_t.second)%60
```

```
        inc_t.second = e_s
```

```
        inc_t.minute=t.minute+e_m
```

```
    return inc_t
```

```
t = Time()
```

```
t.hour,t.minute,t.second=11,30,30
```

```
increment(t,180)
```

```
print_time(t)
```

```
t = Time()
```

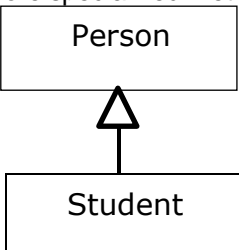
```
t.hour,t.minute,t.second=11,30,30
```

```
inc_t = increment_pure(t,180)
```

```
print_time(inc_t)
```

In the modifier version, the seconds are incremented to the t object. In the pure version of increment, the original t object is not modified, but the seconds incremented to time is returned.

c.	<b>Use the datetime module to write a program that gets the current date and prints the day of the week.</b>	<b>6m</b>
Ans.	import datetime today=datetime.datetime.today() print(today) print(today.weekday())  Output: 2021-03-12 15:19:10.966676	

	<p>4</p> <p>Explanation: from datetime, the today() function returns the current date and time as an object. The weekday() function returns the day of the week where Monday is 0 and Sunday is 6. Because 12.3.2021 is falling on Friday, it returns 4.</p>	
<b>8a.</b>	<b>Explain operator overloading and polymorphism with examples.</b>	<b>8m</b>
Ans.	<p>Operator Overloading</p> <ul style="list-style-type: none"> <li>▶ Changing the <b>behavior of an operator</b> so that it works with user-defined types is called</li> <li>▶ There are special methods that specify behaviour of operators on user defined types</li> <li>▶ <code>__add__</code> will be called if you use the on two objects.</li> </ul> <pre>class Time:     def __init__(self, hour=0,minute=0,second=0):         self.hour=hour         self.minute=minute         self.second=second      def __str__(self):         return '%.2d:%.2d:%.2d'%(self.hour,self.minute,self.second)     def __add__(self, other):         sum = self.time_to_seconds() + other. + operator time_to_seconds()         return self.seconds_to_time(sum)</pre> <pre>t1=Time(2,45,3) t2 = Time(3,15,2) print(t1+t2) # Output 06:00:05</pre> <ul style="list-style-type: none"> <li>▶ Type-based dispatch- not always necessary</li> <li>▶ Functions can be written that work correctly for arguments with different types</li> <li>▶ Functions that can <b>work with several types</b> are called <b>polymorphic</b></li> <li>▶ facilitate code reuse</li> <li>▶ Sum built in function can be used to add because Time can add two objects.</li> <li>▶ <code>print(sum([t1,t2,t3]) )</code></li> </ul>	
<b>b.</b>	<b>Illustrate the concepts of inheritance and class diagrams with examples</b>	<b>8m</b>
Ans.	<p>Inheritance is the ability to define a new class that is a modified version of an existing class. It is called “inheritance” because the new class inherits the methods of the existing class. Extending this metaphor, the existing class is called the <b>parent</b> and the new class is called the <b>child</b>.</p> <p>Inheritance is a useful feature. Some programs that would be repetitive without inheritance can be written more elegantly with it.</p> <p>Inheritance can facilitate code reuse, since you can customize the behavior of parent classes without having to modify them. In some cases, the inheritance structure reflects the natural structure of the problem, which makes the program easier to understand.</p> <p>All the methods in the superclass are accessible to the child class. The child class may define more specialized methods in addition to the methods in the super class.</p> <div style="display: flex; align-items: center;"> <div style="margin-right: 20px;">  <pre> classDiagram     class Person     class Student     Student -- &gt; Person </pre> </div> <div> <p>A <b>class diagram</b> is a graphical representation of these relationships. The arrow with a hollow triangle head represents an IS-A relationship. in this case it indicates that Student inherits from Person.</p> </div> </div>	

	<pre> class Person:     def __init__(self,aadhar_num, name, dob, gender):         self.usn, self.name, self.dob, self.gender = usn, name, dob, gender  class Student(Person):     def __init__(self, aadhar_no, name, dob, gender, usn, branch, year):         super().__init__(aadhar_no,name, dob, gender)         self.usn = usn         self.branch = branch         self.year = year  s1 = Student('87024586','Kumar','07-08-1994','M','CSE', 2014) </pre> <p>The above code snippet demonstrates how Student inherits from Person.</p> <p><b>Explanation:</b></p> <ul style="list-style-type: none"> <li>• The parent class is Person. It has a <code>__init__</code> to instantiate its instance variables, aadhar number, name, date of birth(dob) and gender</li> <li>• A specialization of Person is the Student class which has all the attributes of a Person and in addition, student specific attributes such as USN, branch and year (of admission).</li> <li>• When we instantiate a student object, we may pass all the attributes <code>s1 = Student ('87024586','Kumar','07-08-1994','M','CSE', 2014)</code></li> <li>• When we are instantiating the variables in the Student class, we can invoke the superclass's <code>__init__</code> by using <code>super()</code> to pass the attributes that belong to the super class</li> </ul>	
<b>c.</b>	<b>Write a function called <code>print_time</code> that takes a time object and prints it in the form of hour: minute: second</b>	<b>4m</b>
Ans.	<pre> class Time:     def __init__(self, hour=0,minute=0,second=0):         self.hour=hour         self.minute=minute         self.second=second  def print_time(t):     print( '%.2d:%.2d:%.2d'%(t.hour,t.minute,t.second) )  t1=Time (2,45,3) print_time(t1) </pre> <p>Sample output: 02:45:03</p> <p><b>Explanation:</b> The print function uses <code>%0.2d</code> to add a leading 0 if there is a single digit for hour, minute or second.</p> <p>The dot notation is used to access the attributes of the time object, hour, minute and second respectively.</p>	
<b>MODULE 5</b>		
<b>9 a.</b>	<b>Explain parsing HTML with the BeautifulSoup Module with code snippet for creating finding an element and getting data.</b>	<b>9m</b>
Ans.	<p>Parsing HTML with BeautifulSoup module</p> <ul style="list-style-type: none"> <li>▶ <b>pip install beautifulsoup4</b></li> <li>▶ Download <a href="#">Automate the Boring Stuff with Python, 2nd Edition   No Starch Press</a> and store it as <code>example.html</code></li> </ul>	

- ▶ We'll read from a html page in the hard disk
- ▶ Creating a BeautifulSoup

An object can be created either from a web page or a downloaded page

```
>>> import requests,bs4
>>> res=requests.get('http://nostarch.com')
>>> res.raise_for_status()
>>> soup = bs4.BeautifulSoup(res.text)
```

```
>>> import bs4
>>> fh = open('example.html')
>>> soup = bs4.BeautifulSoup(fh.read())
```

Selector	match
soup.select('div')	All elements named <div>
soup.select('#author')	The element with an id attribute of author
soup.select('.notice')	All elements that use a CSS class attribute named notice
soup.select('div span')	All elements named <span> that are within an element named <div>
soup.select('div > span')	All elements named <span> that are <i>directly</i> within an element named <div>, with no other element in between
soup.select('input[name]')	All elements named <input> that have a name attribute with any value
soup.select('input[type="button"]')	All elements named <input> that have an attribute named type with value button

Let sample.html be

```
<html>
<head> <title> Sample page </title>
</head>

<body>
  <h1> Welcome </h1>

  <p id = 'one'> First paragraph </p>
</body>

</html>
```

```
import bs4
fh = open('sample.html')
soup = bs4.BeautifulSoup(fh.read())
elems = soup.select('#one')
print(elems[0].getText())
```

output:  
First Paragraph

In the above code snippet, the element is selected by ID. The `soup.select()` is used to find the element. We can then extract the attributes of the element. The `getText()` method will extract the text within the `p` tag.

**b. What methods do Selenium's web element object have for simulating mouse clicks and keyboard keys. Explain with python code snippet.**

**6m**

Ans

Clicking Elements:

- ▶ method can be used to follow a **link**
- ▶ make a **selection** on a radio button
- ▶ click a **Submit** button
- ▶ trigger anything that happens when an element is clicked by the mouse.

```
from selenium import webdriver
```

```
browser =
```

```
webdriver.Chrome(executable_path=r"C:\Users\Admin\Downloads\chromedriver.exe"
```

```
)
```

```
browser.get('https://www.crummy.com/software/BeautifulSoup/bs4/doc/')
```

```
try:
```

```
    linkElem = browser.find_element_by_link_text('Sphinx')
```

```
    print(type(linkElem))
```

```
    linkElem.click()
```

```
except:
```

```
    print('Was not able to find an element with that name.')
```

- ▶ These keys are stored in `selenium.webdriver.common.keys` module.

- ▶ run from `selenium.webdriver.common.keys` import `Keys`

```
from selenium import webdriver
```

```
from selenium.webdriver.common.keys import Keys
```

```
browser =
```

```
webdriver.Chrome(executable_path=r"C:\Users\Admin\Downloads\chromedriver.exe"
```

```
)
```

```
browser.get('https://www.crummy.com/software/BeautifulSoup/bs4/doc/')
```

```
try:
```

```
    htmlElem = browser.find_element_by_tag_name('html')
```

	<pre>#Go to the end htmlElem.send_keys(Keys.END) htmlElem.send_keys(Keys.HOME) except: print('Was not able to find an element with that name.')</pre>	
<b>c.</b>	<b>Write a python program to access a cell in a worksheet.</b>	<b>5m</b>
Ans.	<pre>import openpyxl wb= openpyxl.open('ex_excel.xlsx') sheet= wb.active # access using coordinates Cell_A1 = sheet['A1']  # to access the value print(Cell_A1.value)  # access using row and column sheet.cell(row=1,column=2).value</pre> <p>Explanation:</p> <ul style="list-style-type: none"> <li>▶ First we import openpyxl package.</li> <li>▶ It needs to be installed using pip install openpyxl. It works on files from Microsoft Excel or free alternatives like LibreOffice Calc, OpenOffice Calc work with .xlsx files</li> <li>▶ <b>spreadsheet</b> document called workbook</li> <li>▶ openpyxl.open() returns an object of type workbook which we save in <b>wb</b></li> <li>▶ Each workbook can have <b>multiple sheets</b>.</li> <li>▶ The sheet that is currently being viewed is called the <b>active sheet</b>.</li> <li>▶ To obtain a reference to the active sheet we use wb.active.</li> <li>▶ A sheet can be directly accessed using the name also such as, wb['Sheet1']</li> <li>▶ Cell object has a value attribute that contains value</li> <li>▶ Also has <b>row, column, and coordinate</b> attributes</li> <li>▶ To access using coordinate, we can specify the column letter and the row number that starts from 1.</li> <li>▶ sheet['A1'] returns the cell in the first row and first column. Sheet['C3'] returns cell in the 3<sup>rd</sup> column and 3<sup>rd</sup> row.</li> <li>▶ Specifying column by letter can be difficult : because after Z it becomes AA, AB, and so on.</li> <li>▶ <b>cell()</b> method allows passing a number.</li> <li>▶ sheet.cell(row=1,column=2) allows us to access a cell using row number and column number.</li> </ul>	
<b>10a.</b>	<b>Write a program to get a list of all files with the .pdf extension in the current working directory and sort them.</b>	<b>6m</b>
Ans.	<pre>import os,re lstFiles = os.listdir() regexpdf=re.compile('\.pdf\$') pdf_files = [] for file in lstFiles:     if regexpdf.search(file):         pdf_files.append(file)  pdf_files.sort() print(pdf_files)</pre> <p>Sample Output:</p> <pre>['LP.pdf', 'Syll.pdf', 'Syll_protected.pdf', 'VTUTT.pdf', 'conf.pdf', 'coverletter-netapp.pdf', 'encrypted.pdf', 'marked.pdf', 'merged.pdf', 'rotated.pdf', 'watermark.pdf']</pre>	

	<p>Explanation</p> <ul style="list-style-type: none"> <li>• We will import 2 packages, os and re.</li> <li>• <b>os.listdir(path)</b> : returns a list of filename strings for each file in the <i>path</i> argument</li> <li>• If we do not pass any argument to listdir() it returns list of filename strings in the current working directory.</li> <li>• We write a regular expression for files ending with .pdf. Since the dot is a special character in a regular expression, we escape the special meaning to match a literal dot in the file name. next it matches pdf at the end of the string.</li> <li>• A for loop is set up to iterate through the file names from os.listdir(). Each file name is checked with the patter in regexprdf variable. If the search is successful, then the file name string is added to a list pdf_files.</li> <li>• The contents of pdf_files is sorted by invoking sort()</li> <li>• Since lists are mutable, the sort() is done in place.</li> <li>• The sorted list is then printed.</li> </ul>	
<b>b.</b>	<b>Demonstrate the json module with the python program.</b>	<b>6m</b>
Ans.	<ul style="list-style-type: none"> <li>▶ handles all the details of translating between a string with JSON data and Python values for the <b>json.loads()</b> and <b>json.dumps()</b></li> <li>▶ JSON can't store every kind of Python value</li> <li>▶ Following data types are possible : <b>strings, integers, floats, Booleans, lists, dictionaries, and NoneType.</b></li> <li>▶ JSON <b>cannot represent Python-specific objects</b> <ul style="list-style-type: none"> <li>▶ File objects, CSV Reader or Writer objects, Regex objects, or Selenium WebElement objects.</li> </ul> </li> <li>▶ <b>json.loads():</b> translate a string containing JSON data into a Python value <ul style="list-style-type: none"> <li>▶ <b>loads: load string</b></li> </ul> </li> <li>▶ JSON strings always use <b>double quotes</b></li> <li>▶ It returns a <b>Python dictionary.</b> <ul style="list-style-type: none"> <li>▶ Dictionary is not ordered</li> <li>▶ So key value pair may appear in different order.</li> </ul> </li> </ul> <pre>&gt;&gt;&gt; str = '{"TeamID":"CS003", "Topic":"Review 1","Number of attendees":5,"Meet Link":null, "isStarted":false}' &gt;&gt;&gt; import json &gt;&gt;&gt; val = json.loads(str) &gt;&gt;&gt; val {'TeamID': 'CS003', 'Topic': 'Review 1', 'Number of attendees': 5, 'Meet Link': None, 'isStarted': False} ▶ <b>json.dumps()</b> function : translate a Python value into a string of JSON-formatted data ▶ <b>dumps : dump string</b> ▶ value can only be one of the following <b>basic Python data types:</b> ▶ <b>dictionary, list, integer, float, string, Boolean, or None.</b> &gt;&gt;&gt; d={'TeamID': 'CS003', 'Topic': 'Review 1', 'Number of attendees': 5, 'Meet Link': None, 'isStarted': False} &gt;&gt;&gt; str = json.dumps(d) &gt;&gt;&gt; str '{"TeamID": "CS003", "Topic": "Review 1", "Number of attendees": 5, "Meet Link": null, "isStarted": false}'</pre>	
<b>c.</b>	<b>What are the advantages of CSV file? Explain the Reader objects and Writer objects with python code.</b>	<b>8m</b>
Ans.	<ul style="list-style-type: none"> <li>▶ Each line in a CSV file represents a <b>row</b> in the spreadsheet.</li> <li>▶ <b>Commas</b> separate the cells in the row</li> <li>▶ CSV files are <b>simple</b>, they don't have many features of an excel spreadsheet. <ul style="list-style-type: none"> <li>▶ <b>Don't have types</b> for their values—everything is a string</li> <li>▶ <b>Don't</b> have settings for <b>font size or color</b></li> <li>▶ <b>Don't</b> have <b>multiple worksheets</b></li> <li>▶ <b>Can't</b> specify cell <b>widths</b> and <b>heights</b></li> <li>▶ <b>Can't</b> have <b>merged cells</b></li> <li>▶ <b>Can't</b> have <b>images</b> or <b>charts</b> embedded in them</li> </ul> </li> </ul>	

## Advantage of CSV

- ▶ **Simplicity**
- ▶ widely **supported** by many types of programs
- ▶ can be **viewed in text editors** (including IDLE's editor)
- ▶ straightforward way to **represent spreadsheet data**.
- ▶ CSV files also have their **own set of escape characters** to allow commas and other characters to be included *as part of the values*
  - ▶ ***split()* method does not handle these special characters.**
  - ▶ Use **csv module**
  - ▶ **Avoid opening as text file and use split().**

## Reader() function

- ▶ **csv.reader()** function
- ▶ This returns a Reader object
- ▶ **You can't directly pass a filename to the reader() function**
- ▶ Notice that we get a list of lists when we convert the data to a list.

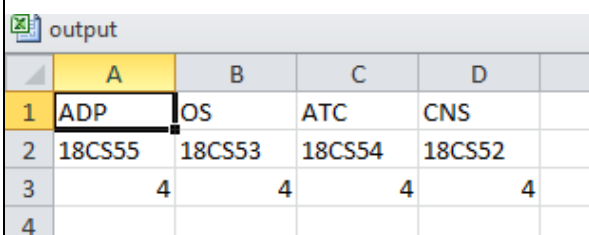
```
>>> import csv
>>> import os
>>> os.chdir('Documents')
>>> fh = open('sample_csv.csv')
>>> reader = csv.reader(fh)
>>> data = list(reader)
>>> data
[['4/5/2015 1:34:02 PM ', 'Apples', '73'], ['4/5/2015 1:34:02 PM ', 'Cherries',...
>>> data[0][1]
'Apples'
>>> data[1][0]
'4/5/2015 1:34:02 PM '
```

- ▶ For large CSV files, use **a for loop**.
- ▶ This avoids loading the entire file into memory at once.
- ▶ The reader can be looped **over once**.
- ▶ To read again, open the file again and obtain a reader object.

```
>>> fh = open('sample_csv.csv')
>>> reader = csv.reader(fh)
>>> for row in reader:
...     print('Row # %d %s'%(reader.line_num,row))
...
Row #1 ['4/5/2015 1:34:02 PM ', 'Apples', '73']
Row #2 ['4/5/2015 1:34:02 PM ', 'Cherries', '85']
....
```

## Writer object

- ▶ A **Writer object** lets you write data to a CSV file.



	A	B	C	D
1	ADP	OS	ATC	CNS
2	18CS55	18CS53	18CS54	18CS52
3	4	4	4	4
4				

- ▶ pass it **'w'** to open a file in write mode
- ▶ pass to **csv.writer()** - creates a writer object
- ▶ In Windows, pass **empty string as newline character** otherwise output will be double spaced.
- ▶ **writerow()** takes a **list** argument

```
>>> of = open('output.csv', 'w', newline='')
>>> opwriter = csv.writer(of)
>>> opwriter.writerow(['ADP','OS','ATC','CNS'])
16
>>> opwriter.writerow(['18CS55','18CS53','18CS54','18CS52'])
29
>>> opwriter.writerow([4,4,4,4])
9
>>> of.close()
```



- ▶ separate cells with a tab character instead of a comma
- ▶ rows to be double-spaced
- ▶ **delimiter**
  - ▶ character that appears **between cells** on a row
- ▶ **line terminator**
  - ▶ character that comes at the **end of a row**

```
>>> of = open('example.csv', 'w', newline='')
>>> opwriter = csv.writer(of, delimiter='\t', lineterminator='\n\n')
>>> opwriter.writerow(['Apples', 'Oranges', 'Lemons'])
23
>>> opwriter.writerow(['45', '50', '30'])
10
>>> of.close()
```

