1.a.# Python Program to find Diameter, Circumference, and Area Of a Circle

```
PI = 3.14
radius = float(input(' Please Enter the radius of a circle: '))

diameter = 2 * radius
circumference = 2 * PI * radius
area = PI * radius * radius

print(" \nDiameter Of a Circle = %.2f" %diameter)
print(" Circumference Of a Circle = %.2f" %circumference)
print(" Area Of a Circle = %.2f" %area)
```

1.b.Python range() function returns the sequence of the given number between the given range.
range() is a built-in function of Python. It is used when a user needs to perform an action a specific number of times

```
# Python Program to
# show range() basics

# printing a number
for i in range(10):
    print(i, end=" ")
print()

# using range for iteration
l = [10, 20, 30, 40]
for i in range(len(l)):
    print(l[i], end=" ")
print()

# performing sum of natural
# number
sum = 0
for i in range(1, 11):
    sum = sum + i
print("Sum of first 10 natural number :", sum)
```

There are three ways you can call range() :

- range(stop) takes one argument.
- range(start, stop) takes two arguments.
- range(start, stop, step) takes three arguments.

c.

Ans:

Parameters and variables that are assigned in a called function are said to exist in that function's local scope. Variables that are assigned outside all functions are said to exist in the global scope.

 Variables that are assigned outside all functions are said to exist in the global scope & they can be modified/corrupted by any of the function.

 A variable must be one or the other; it cannot be both local and global.

 A local scope is created whenever a function is called. Any variables assigned in this function exist within the local scope

 If we ever want to modify the value stored in a global variable from in a function, we must use a global statement on that variable.

Example

```
def func1():
global name
name="Radha"
print("Here name in func1 is "+name);
name='kris'
print("Here name outside of functions is "+name);
func1()
```

```
print("Here name outside of functions is "+name);
```

2.a.break Statements

There is a shortcut to getting the program execution to break out of a while

clauseearly. If the execution reaches a break statement, it immediately exits the while clause. In code, a break statement simply contains the break keyword.

Pretty simple, right? a program that does the same thing as the previousprogram, but it uses a break statement to escape the loop. Enter the following code, and save the file as *yourName2.py*:

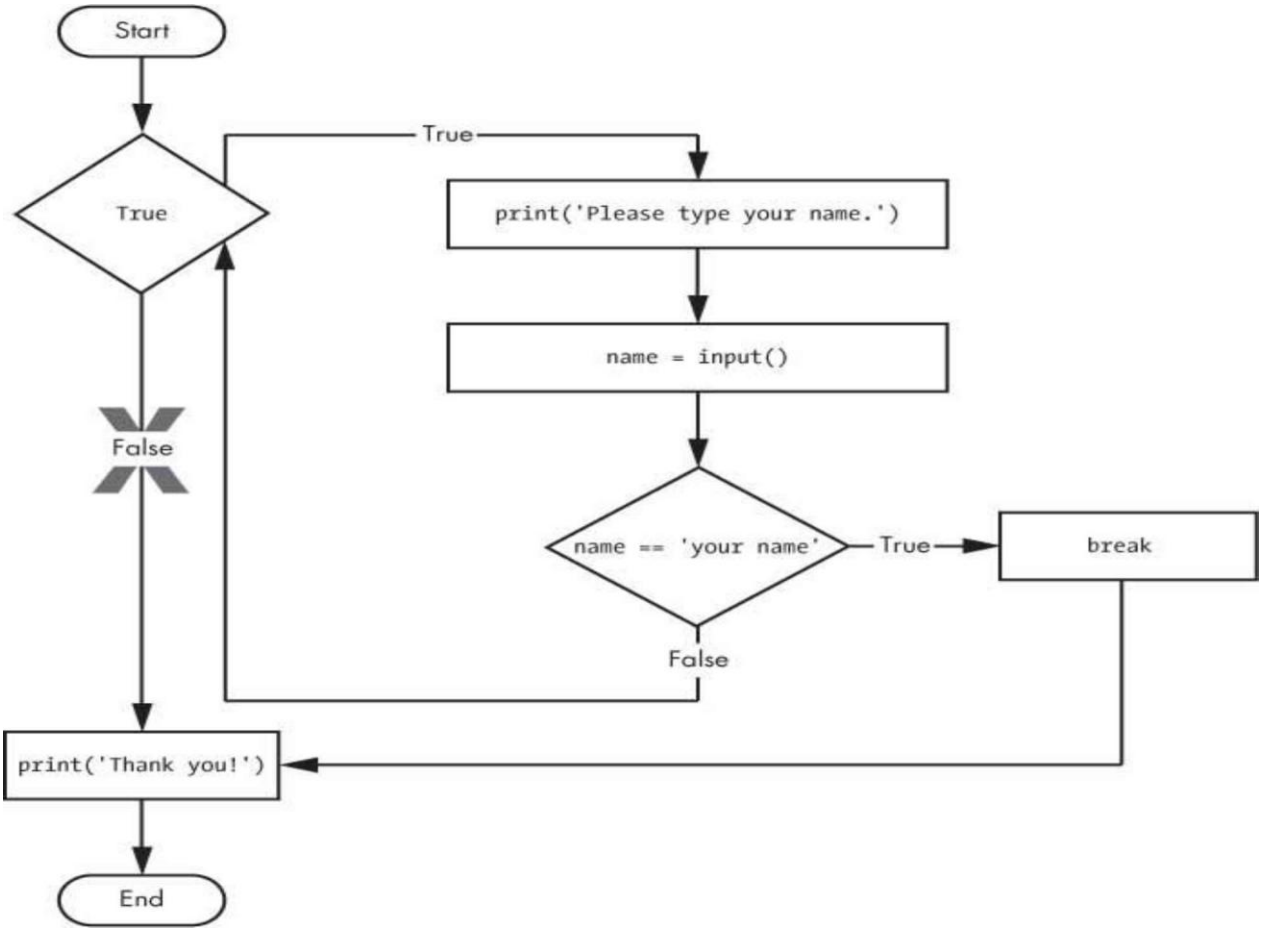
```
while True:
 print('Please type
  your name.') name
  = input()

  if name ==
    'your name':
    break

print('Thank you!')
```

continue Statements

Like break statements, continue statements are used inside loops. When the program execution reaches a continue statement, the program execution immediately jumps back to the start of the loop and reevaluates the condition.



loop's

Let's use

2.b

Operator	Operation	Example	Evaluates to . . .
**	Exponent	2 ** 3	8
%	Modulus/remainder	22 % 8	6

//	Integer division/floored quotient	22 // 8	2
/	Division	22 / 8	2.75
*	Multiplication	3 * 5	15
-	Subtraction	5 - 2	3
+	Addition	2 + 2	4

2.c# Python program to check if

```
# given number is prime or not

num = 11

# If given number is greater than 1

if num > 1:

   # Iterate from 2 to n / 2

   for i in range(2, int(num/2)+1):

      # If num is divisible by any number between

      # 2 and n / 2, it is not prime

      if (num % i) == 0:

         print(num, "is not a prime number")

         break
```

```
    else:
        print(num, "is a prime number")
else:
    print(num, "is not a prime number")
```

Output

11 is a prime number

3a. *Removing Values from Lists with del Statements*

The del statement will delete values at an index in a list. All of the values in the list after the deleted value will be moved up one index. For example, enter the following into the /

interactive shell:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat']
```

The del statement can also be used on a simple variable to delete it, as if it were an "unassignment" statement. If you try to use the variable after deleting it, you will get a NameError error because the variable no longer exists. In practice, you almost never need to delete simple variables. The del

statement is mostly used to delete values from lists.

3b.function to print the longest

```
# word in given sentence

def largestWord(s):

    # Sort the words in increasing

    # order of their lengths

    s = sorted(s, key = len)

    # Print last word

    print(s[-1])

if __name__ == "__main__":

    # Given string

    s = "be confident and be yourself"

    # Split the string into words

    l = list(s.split(" "))

      largestWord(l)
```

Output:

confident

3c. Dictionaries vs. Lists

Unlike lists, items in dictionaries are unordered. The first item in a list named spam would be spam[0]. But there is no "first" item in a dictionary. While the order of items matters for determining whether two lists are the same, it does not matter in what order the key-value pairs are typed in a dictionary. Enter the following into the interactive shell:

```
 spam = ['cats', 'dogs', 'moose']
>>> bacon = ['dogs', 'moose', 'cats']
>>> spam == bacon
False
>>> eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}
>>> ham = {'species': 'cat', 'age': '8', 'name': 'Zophie'}
>>> eggs == ham
True
```

Because dictionaries are not ordered, they can't be sliced like lists.

Trying to access a key that does not exist in a dictionary will result in a KeyError error message, much like a list's "out-of-range" IndexError error message. Enter the following into the interactive shell, and notice the error message that shows up because there is no 'color' key:

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> spam['color']
Traceback (most recent call last):
 File "<pyshell#1>", line 1,
  in <module>
  spam['color']
```

KeyError: 'color'

3b. The isupper() and islower() methods will return a Boolean True value if the string has at least one letter and all the letters are uppercase or lowercase, respectively. Otherwise, the method returns False. Enter the following into the interactive shell, and notice what each method call returns:

```
>>> spam = 'Hello, world!'
>>> spam.islower()
False
>>> spam.isupper()
False
>>> 'HELLO'.isupper()
True
>>> 'abc12345'.islower()
True
>>> '12345'.islower()
False
>>> ', '.join(['cats', 'rats', 'bats'])
'cats, rats, bats'
>>> ' '.join(['My', 'name', 'is', 'Simon'])
'My name is Simon'
>>> 'ABC'.join(['My', 'name', 'is', 'Simon'])
'MyABCnameABCisABCSimon'
>>> 'My name is Simon'.split()
['My', 'name', 'is', 'Simon']
```

By default, the string 'My name is Simon' is split wherever whitespace characters such as the space, tab, or newline characters are found. These whitespace characters are not included in the strings in the returned list. You can pass a delimiter string to the split() method to specify a different string to split upon. For example, enter the following into the interactive shell:

```
>>> '12345'.isupper()
False
```

```
4ab. num_list=[]
n=int(input("Enter the Starting of the range:"))
k=int(input("Enter the Ending of the range:"))
for i in range(n,k):
num_list.append(i)
print("Original Number List:", num_list)
even_list=[]
odd_list=[]
for i in range(len(num_list)):
if(num_list[i]%2==0):
even_list.append(num_list[i])
else:
odd_list.append(num_list[i])
print("Even Numbers List:", even_list)
print("Odd Numbers List:", odd_list)
```

Output:

```
Enter the Starting of the range:5
Enter the Ending of the range:15
Original Number List: [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
Even Numbers List: [6, 8, 10, 12, 14]
Odd Numbers List: [5, 7, 9, 11, 13]
>>>
```

4c. **>>> spam = 'Hello, world!'**

>>> spam[0]

'H'

```
>>> spam[4]
```

'o'

```
>>> spam[-1]
```

'!'

```
>>> spam[0:5]
```

'Hello'

```
>>> spam[:5]
```

'Hello'

```
>>> spam[7:]
```

'world!'

5a. While there are several steps to using regular expressions in Python, each step is fairly simple.

1. Import the regex module with import re.
2. Create a Regex object with the re.compile() function. (Remember to use a raw string.)
3. Pass the string you want to search into the Regex object's search() method. Thisreturns a Match object.
4. Call the Match object's group() method to return a string of the actual matched text.

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> print('Phone number found: ' + mo.group())
```

Phone number found: 415-555-4242

5b. *Grouping with Parentheses*

Say you want to separate the area code from the rest of the phone number. Adding parentheses will create *groups* in the regex: (\d\d\d)-(\d\d\d-\d\d\d\d). Then you can use the group() match object method to grab the matching text from just one group.

The first set of parentheses in a regex string will be group 1. The second set will be group 2. By passing the *integer* 1 or 2 to the group() match object method, you can grab different parts of the matched text. Passing 0 or nothing to the group() method will return the entire matched text. Enter the following into the interactive shell:

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> mo.group(1)
'415'
>>> mo.group(2)
'555-4242'
>>> mo.group(0)
'415-555-4242'
>>> mo.group()
'415-555-4242'
```

If you would like to retrieve all the groups at once, use the groups() method—note the plural form for the name.

```
>>> mo.groups()
('415', '555-4242')
>>> areaCode, mainNumber = mo.groups()
>>> print(areaCode)
415
>>> print(mainNumber)
555-4242
```

Since mo.groups() returns a tuple of multiple values, you can use the multiple- assignment trick to assign each value to a separate variable, as in the previous areaCode, mainNumber = mo.groups() line.

Parentheses have a special meaning in regular expressions, but what do you do if you need to match a parenthesis in your text? For instance, maybe the phone numbers you are trying to match have the area code set in parentheses. In this case, you need to escape the(and) characters with a backslash. Enter the following into the interactive shell:

```
>>> phoneNumRegex = re.compile(r'(\(\d\d\d\)) (\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My phone number is (415) 555-4242.')
>>> mo.group(1)
'(415)'
>>> mo.group(2)
'555-4242'
```

The \(and \) escape characters in the raw string passed to re.compile() will match actual parenthesis characters. In regular expressions, the following

characters have special meanings:

. ^ $ * + ? { } [ ] \ | ( )

If you want to detect these characters as part of your text pattern, you need to escape them with a backslash:

\. \^ \$ * \+ \? \{ \} \[\] \\ \| \(\)

Make sure to double-check that you haven't mistaken escaped parentheses \(and \) for parentheses (and) in a regular expression. If you receive an error message about "missing)" or "unbalanced parenthesis," you may have forgotten to include the closing unescaped parenthesis for a group, like in this example:

```
>>> re.compile(r'(\(Parentheses\)')
Traceback (most recent call last):
  --snip--
re.error: missing ), unterminated subpattern at position 0
```

The error message tells you that there is an opening parenthesis at index 0 of the r'(\ (Parentheses\)' string that is missing its corresponding closing parenthesis.

Matching Multiple Groups with the Pipe

The | character is called a *pipe*. You can use it anywhere you want to match one of many expressions. For example, the regular expression r'Batman|Tina Fey' will match either 'Batman' or 'Tina Fey'.

/

hen *both* Batman and Tina Fey occur in the searched string, the first

occurrence of matching text will be returned as the Match object. Enter the following into the interactive shell:

```
>>> heroRegex = re.compile (r'Batman|Tina Fey')
>>> mo1 = heroRegex.search('Batman and Tina Fey')
>>> mo1.group()
'Batman'

>>> mo2 = heroRegex.search('Tina Fey and Batman')
>>> mo2.group()
'Tina Fey'
```

5c. *Copying Files and Folders*

The shutil module provides functions for copying files, as well as entire folders.

Calling shutil.copy(*source*, *destination*) will copy the file at the path *source* to the folder at the path *destination*. (Both *source* and *destination* can be strings or Path objects.) If *destination* is a filename, it will be used as the new name of the copied file. This function returns a string or Path object of the copied file.

Enter the following into the interactive shell to see how shutil.copy() works:

```
>>> import shutil, os
>>> from pathlib import Path
```

```
>>> p = Path.home()
❶ >>> shutil.copy(p / 'spam.txt', p / 'some_folder')
'C:\\Users\\Al\\some_folder\\spam.txt'
```

Moving and Renaming Files and Folders

Calling shutil.move(*source*, *destination*) will move the file or folder at the path *source* to thepath *destination* and will return a string of the absolute path of the new location.

If *destination* points to a folder, the *source* file gets moved into *destination* and keeps its current filename. For example, enter the following into the interactive shell:

```
>>> import shutil
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs\\bacon.txt'
```

Permanently Deleting Files and Folders

You can delete a single file or a single empty folder with functions in the os module, whereas to delete a folder and all of its contents, you use the shutil module.

Calling os.unlink(*path*) will delete the file at *path*.

Calling os.rmdir(*path*) will delete the folder at *path*. This folder must be empty of anyfiles or folders.

Calling shutil.rmtree(*path*) will remove the folder at *path*, and all files and folders itcontains will also be deleted.

6a,b,c

COMPRESSING FILES WITH THE ZIPFILE MODULE

You may be familiar with ZIP files (with the *.zip* file extension), which can hold the compressed contents of many other files. Compressing a file reduces its size, which is useful when transferring it over the internet. And since a ZIP file can also contain multiple files and subfolders, it's a handy way to package several files into one. This single file, called an *archive file*, can then be, say, attached to an email.

Your Python programs can create and open (or *extract*) ZIP files using functions in the zipfile module. Say you have a ZIP file named *example.zip*

Reading ZIP Files

To read the contents of a ZIP file, first you must create a ZipFile object (note the capital letters *Z* and *F*). ZipFile objects are conceptually similar to the File objects you saw returned by the open() function in the previous chapter: they are values through which the program interacts with the file. To create a ZipFile object, call the zipfile.ZipFile() function, passing it a string of the *.ZIP* file's filename. Note that zipfile is the name of the Python module, and ZipFile() is the name of the function.

For example, enter the following into the interactive shell:

```
>>> import zipfile, os
>>> from pathlib import Path
>>> p = Path.home()
>>> exampleZip = zipfile.ZipFile(p / 'example.zip')
>>> exampleZip.namelist()
['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
>>> spamInfo = exampleZip.getinfo('spam.txt')
>>> spamInfo.file_size
13908
```

```
>>> spamInfo.compress_size
3828
❶ >>> f'Compressed file is {round(spamInfo.file_size / spamInfo
.compress_size, 2)}x smaller!'
)
'Compressed file is 3.63x smaller!'
>>> exampleZip.close()
```

A ZipFile object has a namelist() method that returns a list of strings for all the files and folders contained in the ZIP file. These strings can be passed to the getinfo() ZipFile method to return a ZipInfo object about that particular file. ZipInfo objects have their own attributes, such as file_size and compress_size in bytes, which hold integers of the original file size and compressed file size, respectively. While a ZipFile object represents an entire archive file, a ZipInfo object holds useful information about a *single file* in the archive.

The command at ❶ calculates how efficiently *example.zip* is compressed by dividing the original file size by the compressed file size and prints this information.

Extracting from ZIP Files

The extractall() method for ZipFile objects extracts all the files and folders from a ZIP file into the current working directory.

```
>>> import zipfile, os
>>> from pathlib import Path
>>> p = Path.home()
>>> exampleZip = zipfile.ZipFile(p / 'example.zip')
```

❶ >>> exampleZip.extractall()

>>> exampleZip.close()

>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')

>>> mo = phoneNumRegex.search('Cell: 415-555-9999 Work: 212-555-0000')

>>> mo.group()

'415-555-9999'

>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') # has no groups

>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')

['415-555-9999', '212-555-0000']

7 a,b,c

- **A complete program:**
- Write a class Point representing a point on coordinate system. Implement following functions –
  - A function read_point() to receive x and y attributes of a Point object as user input.
  - A function distance() which takes two objects of Point class as arguments and computes the Euclidean distance between them.
    - A function print_point() to display one point in the form of ordered-pair.

Program:

```
#function to read the attributes of the Point object
def read_point(p):
  p.x=float(input
  ("x
  coordinate:"))
  p.y=float(input
  ("y
  coordinate:"))

#function to print the attributes of the Point object
def print_point(p):
  print("(%g,%g)"%(p.x, p.y))

#function which takes two objects of Point class as arguments and
computes the Euclidean distance between them
def distance(p1,p2):

  d=math.sqrt((p1.x-
  p2.x)**2+(p1.y-p2.y)**2)
  return d

p1=Point()                          #create first object
print("Enter First point:")
read_point(p1)                      #read x and y for p1
```

```
p2=Point()                          #create second object
print("Enter Second point:")
read_point(p2)                      #read x and y for p2

dist=distance(p1,p2)                #compute distance

print("First point is:")
print_point(p1)                     #print p1

print("Second point is:")
print_point(p2)                     #print p2

print("Distance is: %g" % dist)     #print the Euclidean distance
between p1 and p2
```

Type-based dispatch in Python

```
class Time:
  """Represents the time of day. attributes: hour, minute, second"""


  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 increment(time, seconds):

    time.second += seconds



    if time.second >= 60:

        s = time.second / 60

        time.second -= int(s) * 60

        time.minute += int(s)



    if time.minute >= 60:

        m = time.minute / 60

        time.minute -= int(m) * 60

        time.hour += int(m)

    return time

def add_time(self, other):
```

```
        sum = Time()
        sum.hour = self.hour + other.hour
        sum.minute = self.minute + other.minute
        sum.second = self.second + other.second
        if sum.second >= 60:
            sum.second -= 60
            sum.minute += 1
        if sum.minute >= 60:
            sum.minute -= 60
            sum.hour += 1
        return sum
    def __add__(self, other):
        if isinstance(other, Time):
            return self.add_time(other)
        else:
            return self.increment(other)


print('----------------')
start = Time(9, 45)
```

```
duration = Time(1, 35)

print(start + duration)



print('----------------')

start = Time(9, 45)

duration = Time(1, 35)

print(start + 1337)
```

Output

```
----------------

11:20:00

----------------

10:07:17
```

Pure functions

The function that creates a new Time object, initializes its attributes with new values and returns a reference to the new object. This is called a pure function because it does not modify any of the objects passed to it as arguments and it has no effect, like displaying a value or getting user input, other than returning a value.

Polymorphism means multiple forms. In python we can find the same operator or function taking multiple forms. It also useful in creating different classes which will have class methods with same name.

```
class India():
    def capital(self):
        print("New Delhi is the capital of India.")
```

```
    def language(self):
        print("Hindi is the most widely spoken language of India.")

    def type(self):
        print("India is a developing country.")

class USA():
    def capital(self):
        print("Washington, D.C. is the capital of USA.")

    def language(self):
        print("English is the primary language of USA.")

    def type(self):
        print("USA is a developed country.")

obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
    country.capital()
    country.language()
    country.type()
```

8a,b,c

The init method

The init method (short for "initialization") is a special method that gets invoked when an object or instance is instantiated. Its full name is **__init__** (two underscore characters, followed by init, and then two more underscores). An init method is like a constructor in java or dot net. An init method for the Time class might look like this, in this case whenever an object of Time is created with no parameter the default values are initialized to the hour, minute and second. If you pass parameters accordingly the values are initialized. in time Hour, minute and second are initialized to 0. In time 2 Hours is initialized to 10 but a minute and second are initialized to 0 and so on.

The str method

str is another special method, like **init**, which returns a string representation of an object. For example, here str method for Time objects to return string form of time.

```
class Time:
    """Represents the time of day. attributes: hour, minute, second"""

    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)

print('----------------')
time1 = Time()
print(time1)

print('----------------')
time2 = Time(10, 20)
print(time2)
```

9a,b,c

```
import requests

res =
requests.get('http://www.gutenberg.org/cache/epub/1112/pg1112.tx
t') res.raise_for_status()

playFile = open('RomeoAndJuliet.txt', 'wb')

for chunk in res.iter_content(100000):
    playFile.write(chunk)
```

Getting Cells from the Sheets

- Once we have a Worksheet object, we can access a Cell object by its name.
- The Cell object has a value attribute that contains value stored in that cell.
- Cell objects also have row, column, and coordinate attributes that provide location information for the cell.

- Using the sheet's cell() method and passing it row=1 and column=2 gets a Cell object for cell B1, just like specifying sheet['B1'] did.

In [31]:
```
import openpyxl

#new workbook object contaning only one sheet with 'Sheet'

wb = openpyxl.Workbook()
```

```
['Sheet']
```

In [32]:
```
sheet = wb.get_active_sheet()
sheet.title = 'V ADP'

print(wb.get_sheet_names())
```

```
['V ADP']
```

In [33]:
```
import openpyxl

wb = openpyxl.Workbook()
print("The sheets are:")
print(wb.get_sheet_names())
sheet = wb.get_active_sheet()
sheet.title = 'V ADP'
print("The sheets are:")
```

```
The
sheet
s are:
['She
et']
The
sheets
are:
['V
ADP']
```

[36]:
```
import openpyxl

wb = openpyxl.Workbook()

print(wb.get_sheet_names())
wb.create_sheet()

wb.create_sheet(index=0, title='First Sheet')
wb.create_sheet(index=2, title='Middle Sheet')
print(wb.get_sheet_names())

wb.remove_sheet(wb.get_sheet_by_name('Middle Sheet'))
```

```
['Sheet']

['First Sheet', 'Sheet',
'Middle Sheet', 'Sheet1']
['First Sheet', 'Sheet']
```

In [ ]:
```
wb.remove_sheet(wb.get_sheet_by_name('Middle Sheet'))
wb.remove_sheet(wb.get_sheet_by_name('Sheet1'))

print(wb.get_sheet_names())
wb.save('example_copy.xlsx')
```

In [37]:
```
import openpyxl

wb = openpyxl.Workbook()

sheet = wb.get_sheet_by_name('Sheet')
sheet['A1'] = 'Hello world!'

sheet['B1'] = 435
```

10 a,b,c

In [ ]:
```
import PyPDF2
```

In [ ]:
```
# importing required modules

import PyPDF2



# creating a pdf file object

pdfFileObj = open('combinedminutes.pdf', 'rb')



# creating a pdf file reader object, pdfReader

pdfReader = PyPDF2.PdfFileReader(pdfFileObj) printing number of pages in pdf file

# extract author name, file name, file size

print(pdfReader.numPages)
```

In [ ]:
```
import os
os.getcwd()
```

In [ ]:
```
#Program to decrypt PDF file

import PyPDF2



# creating a pdf file reader object, pdfReader

pdfFile=open('encrypted.pdf', 'rb')
```

In [ ]:
```
# creating an object of page object clas

pageObj=pdfReader.getPage(0)
```

In [ ]:
```
#usage of decrpyt method, it returns 1 if the password is correct,
#else 0 if incorrect

pdfReader.decrypt('rosebud')
```

In [ ]:
```
# creating an object of page object class

pageObj = pdfReader.getPage(0)

# extracting text from page
```

Encrypting Pdf's:

- A PdfFileWriter object can also add encryption to a PDF document.
- Before calling the write() method to save to a file, call the encrypt() method and pass it a password string u. PDFs can have a user password (allowing you to view the PDF) and an owner password (allowing you to set permissions for printing, commenting, extracting text, and other features).
- The user password and owner password are the first and second arguments to encrypt(), respectively.
- If only one string argument is passed to encrypt(), it will be used for both passwords.
- In this example, we copied the pages of meetingminutes.pdf to a PdfFileWriter object. We encrypted the PdfFileWriter with the password swordfish, opened a new PDF called encryptedminutes.pdf, and wrote the contents of the PdfFileWriter to the new PDF.
- Before anyone can view encryptedminutes.pdf, they'll have to enter this password.
- **PROGRAM:**

```
#Program to create new PDF file by copying pages of existing PDF's

import PyPDF2

#opening files in read-binary mode

pdf1File = open('lp.pdf', 'rb') pdf2File
= open('syll.pdf', 'rb')

#creating PdfFileReader objects of two
pdf's pdf1Reader =
PyPDF2.PdfFileReader(pdf1File) pdf2Reader =
PyPDF2.PdfFileReader(pdf2File)

#creating object of class PdfFileWriter

pdfWriter = PyPDF2.PdfFileWriter()


for pageNum in range(pdf1Reader.numPages): # 19 pages-- 0 to 18

    pageObj = pdf1Reader.getPage(pageNum) # page index starts from
    0 to

    pdfWriter.addPage(pageObj) #adds page at the end to the
    pdfWriter


for pageNum in range(pdf2Reader.numPages): #21 pages --- 0 to 20

    pageObj = pdf2Reader.getPage(pageNum)

    pdfWriter.addPage(pageObj) #19+21 = 40 pages

pdfOutputFile = open('adp.pdf', 'wb')

#use write() method
```

```
pdfWriter.write(pdfOutputFile)


pdfOutputFile.close()
pdf1File.close()
pdf2File.close()
```

- ***Copying pages:***

- We can use PyPDF2 to copy pages from one PDF document to another.
- This allows to combine multiple PDF files, cut unwanted pages, or reorder pages.
- In the below program, Open both PDF files in read binary mode and store the two resulting File objects in pdf1File and pdf2File.
- Call PyPDF2.PdfFileReader() and pass it pdf1File to get a PdfFileReader object for meetingminutes.pdf
- Call it again and pass it pdf2File to get a PdfFileReader object for meetingminutes2.pdf
- Then create a new PdfFileWriter object, which represents a blank PDF document
- Next, copy all the pages from the two source PDFs and add them to the PdfFileWriter object. Get the Page object by calling getPage() on PdfFileReader object. Then pass that Page object to your PdfFileWriter's addPage() method.

These steps are done first for pdf1Reader and then again for pdf2Reader.

JSON stands for JavaScript Object Notation

- JSON is a standard format for data exchange, which is inspired by JavaScript. Generally, JSON is in string or text format for storing and transporting data
- JSON is often used when data is sent from a server to a web page
- JSON is "self-describing" and easy to understand

 JavaScript Object Notation is a popular way to format data as a single human-readable string.

JSON is useful to know, because many websites offer JSON content as a way for programs to interact with the website. This is known as providing an application programming interface (API).

 Accessing an API is the same as accessing any other web page via a URL.

 The difference is that the data returned by an API is formatted (with JSON, for example) for machines; APIs aren't easy for people to read.

 Many websites make their data available in JSON format. Facebook,Twitter, Yahoo, Google, Tumblr, Wikipedia, Flickr, Data.gov, Reddit, IMDb,Rotten Tomatoes, LinkedIn, and many other popular sites offer APIs for programs to use.

 JSON Syntax Rules

    Data is in name/value pairs
    Data is separated by commas
    Curly braces hold objects
    Square brackets hold arrays

Reading JSON with the loads() Function

 To translate a string containing JSON data into a Python value, pass it to the json.loads() function. (The name means —load string,‖ not —loads.‖)

 First import the json module, then call loads() and pass to it a string of JSON data.

 Note that JSON strings always use double quotes. It will return that data as a Python dictionary.

 Python dictionaries are not ordered, so the key-value pairs may appear in a different order when printed

 Example:

```
>>> import json
>>>stringOfJsonData = '{"name": "Zophie", "isCat": true, "miceCaught": 0, "felineIQ": null}'
>>> jsonDataAsPythonValue = json.loads(stringOfJsonData)
>>> jsonDataAsPythonValue
{'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}
```

Writing JSON with the dumps() Function

 The json.dumps() function (which means -dump string, not -dumps) will translate a Python value into a string of JSON-formatted data.

 Example:

```
>>> pythonValue = {'isCat': True, 'miceCaught': 0, 'name': 'Zophie','felineIQ': None}
>>> import json
>>> stringOfJsonData = json.dumps(pythonValue)
>>> stringOfJsonData
'{"isCat": true, "felineIQ": null, "miceCaught": 0, "name": "Zophie" }'
```

- Note that the value can only be one of the following basic Python data types: dictionary, list, integer, float, string, Boolean, or None.