Programming Logic & Design – Python

By

Mr. Ahmed Amer EL-HASHEMITE

IS Lecturer – SITC / UTG

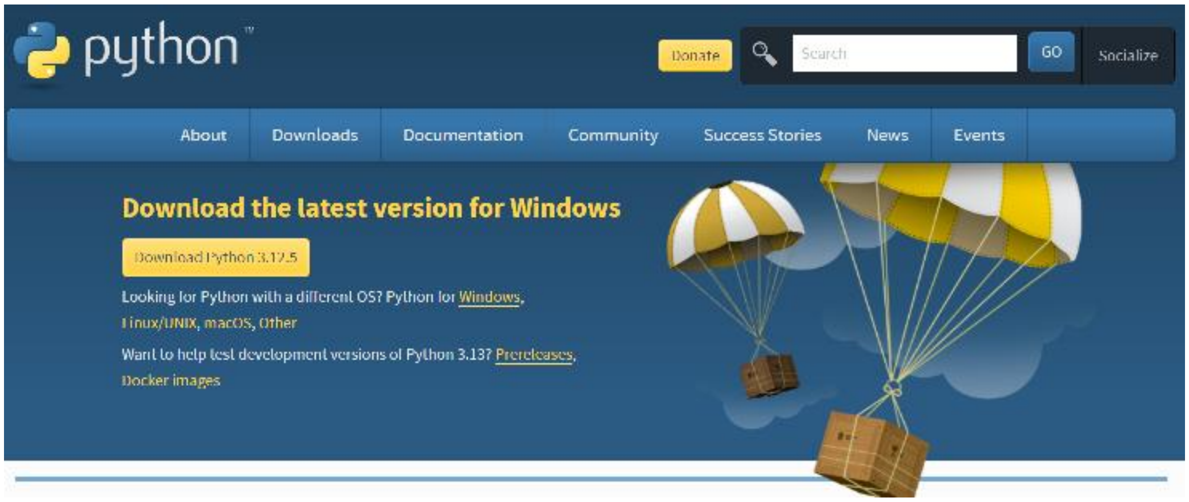
1st Semester 2024-25

FYI

- Each student needs a laptop. The student is responsible for installing the Python Language's Integrated Development and Learning Environment (IDLE) software that is recommended for this course
- This hands-on course is designed to teach students structured programming concepts and problem-solving skills
- By using pseudocode, flowcharts, and python's IDLE, the student will learn to design the logic of programs and translate the pseudocodes / flowcharts into running Python programs
- Student are encouraged to keep up with weekly lectures and do the assigned exercises

Installing & Running Python IDLE

- Connect to the internet
- Open web browser
- Type https://www.python.org/downloads/ in the address bar



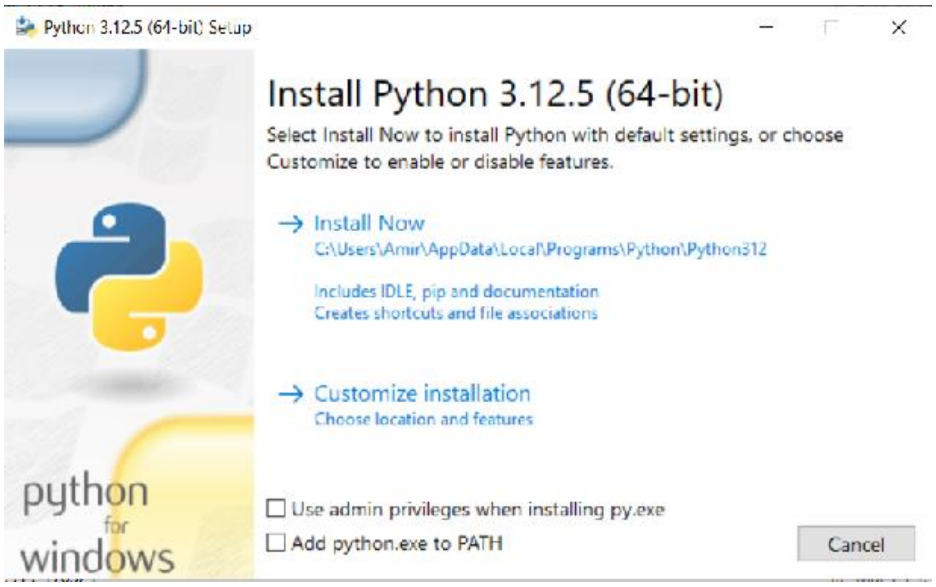
Download the latest version for Windows

Download Python 3.12.5

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python 3.13? Prereleases, Docker images

- Click on the "Download Python 3.12.5" button and download the file python-3.12.5-amd64
- Goto to downloads folder and run the file python-3.12.5-amd64



Install Python 3.12.5 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\Amir\AppData\Local\Programs\Python\Python312

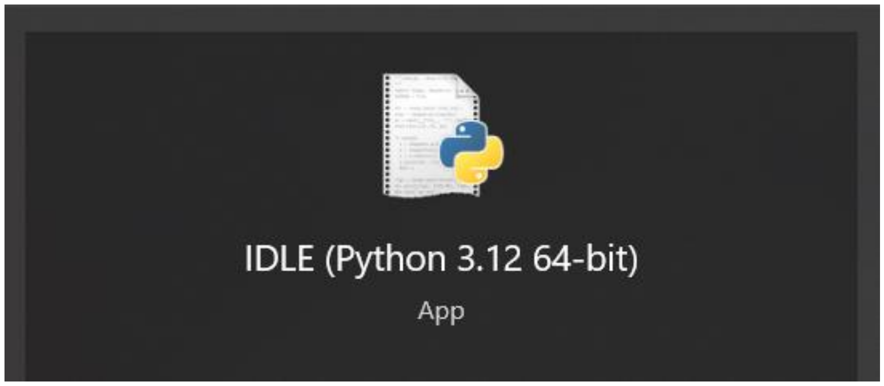
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

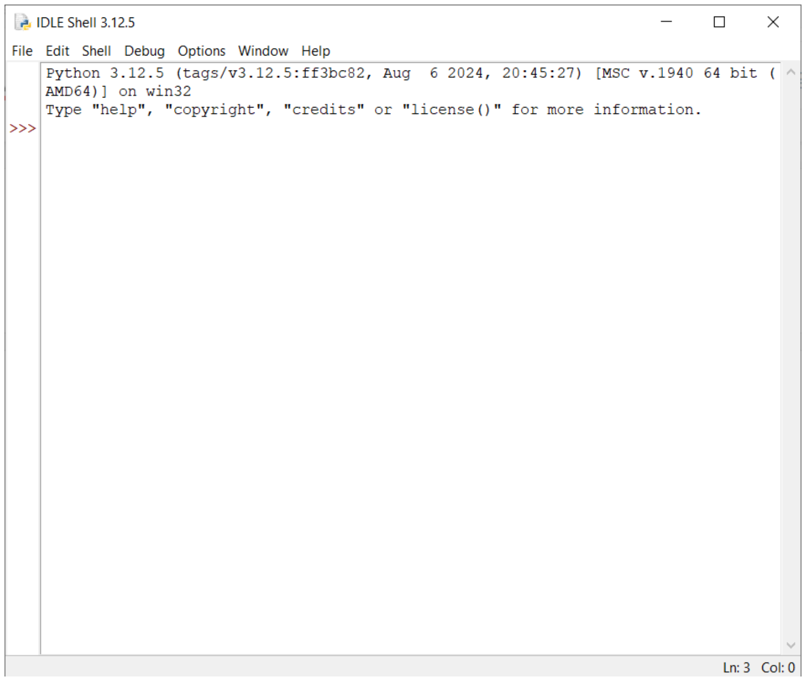
☐ Use admin privileges when installing py.exe
☐ Add python.exe to PATH

- Click on "Install Now" and installing will start
- When installation is completed, click "Close"

- To run Python, type “idle” in the search bar



- Click on the above to open the Python IDLE Shell



```
Python 3.12.5 (tags/v3.12.5:ff3bc82, Aug  6 2024, 20:45:27) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

Introduction

A computer is an electronic device that processes data according to instructions that are provided by computer programs. The computer has two main parts: hardware and software. The hardware is responsible for executing tasks, and the software provides the hardware with directions for what tasks to perform – and how to perform them

The hardware consists of:

- Central Processing Unit (CPU) which runs the programs
- Main Memory (RAM) where a computer stores a program while the program is running, as well as the data that the program is working with – as long as electricity is ON
- Secondary Storage Devices (e.g. Hard Drive, USB, CD / DVD) where data and programs are stored for long periods of time, even when there is no power to the computer
- Input Devices (e.g. Keyboard, Mouse / Trackball , Scanner, Microphone, Digital Camera, HD, CD / DVD, USB)
- Output Devices (e.g. Monitor, Printer, Speakers, HD, CD / DVD, USB)

Electronic devices need electricity, which has two states ON / OFF. This leading us to the Binary system – Machine Language – where all data that is stored in a computer is converted to sequences of 0s and 1s. The computer's memory is divided into Bytes. Each Byte consists of eight bits. Each bit is either a 0 or 1

A computer's CPU can only understand instructions that are written in machine language. Because people find it very difficult to write entire programs in machine language, other programming languages have been invented

The computer languages are divided into:

- Low-Level Language (e.g. Assembly) that is closer to the machine language than human language
- Middle Level Language (e.g. C, C++) that supports the features of both Low-Level and High-Level languages
- High-Level Language (e.g. Java, Python, BASIC, Fortran, COBOL) that is closer the human language

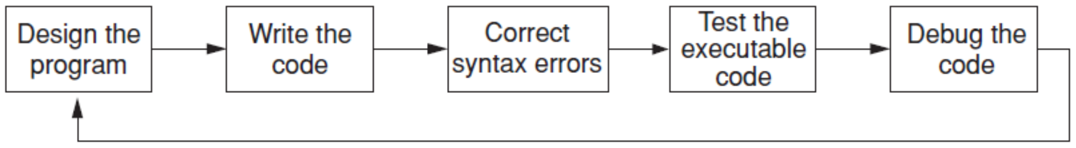
When programmers begin a new project, they never jump right in and start writing code as the first step. They begin by creating a design of the program. After designing the program, the programmer begins writing code in a high-level language.

A language's syntax rules dictate things such as how key words, operators, and punctuation characters can be used. A syntax error occurs if the programmer violates any of these rules.

If the program contains a syntax error, or even a simple mistake such as a misspelled key word, the compiler or interpreter will display an error message indicating what the error is.

Once the code is in an executable form, it is then tested to determine whether any logic errors exist. A logic error is a mistake that does not prevent the program from running but causes it to produce incorrect results.

If there are logic errors, the programmer debugs the code. This means that the programmer finds and corrects the code that is causing the error. Sometimes during this process, the programmer discovers that the original design must be changed. This entire process, which is known as the program development cycle, is repeated until no errors can be found in the program.



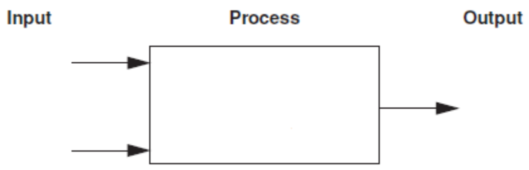
The process of designing a program:

- Understand the task that the program is to perform.

The programmer studies the information that was gathered from the customer during the interviews and creates a list of different software requirements.
A <u>software requirement</u> is simply a single function that the program <u>must perform</u> to satisfy the customer. Once the customer agrees that the list of requirements is complete, the programmer can move to the next phase.

- Determine the steps that must be taken to perform the task.

Once you understand the task that the program will perform, you begin by breaking down the task into a series of logical steps (Algorithm)



Python

❖ Syntax

Remember: Python uses **indentation** to indicate a block of code. Use the same number of spaces in the same block of code, otherwise Python will give you an error

Use **#** to include a line comment that will not affect the code

```
'''
Use ''' to include a block of comments that will not affect the code
'''
```

```
'''
if 5 > 2:
  print("Five is greater than two!")
'''
```

```
'''
```

Syntax Errors

```
if 5 > 2:
print("Five is greater than two!")

if 5 > 2:
    print("Five is greater than two!")
        print("Five is greater than two!")
'''
```

❖ Variables

- Python has no command for declaring a variable
- It is recommended that you declare variables at the beginning of the program (structured programming)
- Every time you declare a new variable, you need to give it an initial value using the assignment sign **=**
- Variables can only include letters (a-z, A-Z), numbers, or the underscore (_)
- Variables are **case-sensitive** (e.g. myvar is not the same as myVar)
- Variables cannot start with a number
- Variables cannot be any of the Python keywords (for, as, with, return, False, True, if, while, None, try, def, del, elif, else …etc.)

❖ Data Types

# Integers
# Float
# String
# Boolean: has two value True of False

❖ Casting

You can use casting to specify the data type of a variable
# int(*argument*)
# str(*argument*)
# float(*argument*)

```
'''
x = 4            # x is of type integer
y = "Sally"      # y is ff type string
'''
```

```
'''
To assign a multiline string to a string variable, use '''

a = ''' Python was designed for readability,
and has some similarities to the English language with influence from mathematics'''
'''
```

```
'''
Python allows to assign value(s) to multiple variables in one line

x, y, z = "Hi", "Red", "Babu"
X = Y = Z = "Babu"
age, name = 30, 'Hawk'
'''
```

```
'''
x = int(5.74566)          # will return 5 – anything after the decimal point is removed
x = int ("Hello")         # will return an error message
x = int ("4.22321")       # will return an error message


y = float("2")            # will return 2.0
y = float("4.5674")       # will return 4.5674
z = str(9.0)              # will return a string '9.0'
'''
```

❖ Basic Operators

```
# +    : Add (can be used to add numbers or strings)
# *    : Multiply two numbers
# -    : Subtract two numbers
# /    : Divide two numbers (the result is a float number)
# //   : floor division – rounds down the answer to the nearest whole number
# %    : gives the remainder of a division (similar to MOD)
# **   : to the power (similar to ^)
# +=   : argument += value (similar to argument = argument + value)
# -=   : argument -= value (similar to argument = argument – value)
# *=   : argument *= value (similar to argument = argument * value)
```

```
'''
x = 10
y = 7
z = 0
w= 15

z = x + y
z = x*y
z = x – y
z = x/y
z = x//y
z = x%y
z = x**y

w += 2
w -= 2
w *= 2
'''
```

```
'''
x = "Babu "
y = "is "
z= "awesome"
w = ""

w = x + y + z                # will return a string 'Babu is awesome'
'''
```

❖ input()

- The input() function takes input from the user and returns it
- Syntax: input('*prompt*'). By default, the function doesn't require any parameters; however, there is one optional parameter – prompt which is a text message to the user
- input() return a **string** value
- To return an integer, use int()
- To return a float, use float()

```
'''
x = input('enter a name: ')              # will return a string assigned to x
y = int(input('enter a number')          # will return an integer number assigned to y
z = float(input('enter a real number')   # will return a float number assigned to z
'''
```

❖ print()

- The print() function prints object(s) to screen or text file
- Syntax: print(object(s), sep='*separator*', end='*type of end*', file='*where to print*')
- object(s): number, character, mathematical process, string
- sep: option to separate 2 or more objects. By default, it is single space ' '
- end: option to end the print. By default, it is newline '\n'
- file: option to where to print. by default, it is the screen
- Use format() to insert numbers into strings. The format() takes **unlimited** number of arguments and are placed into the respective placeholders using index numbers **{***number***}**. Number of arguments starts at 0

```
'''
print('Hi PLD')               # use either ' or " for a string

x=5
y=10
z=x+y

print(x+y)                    # will print the x+y which is 15
print(z)                      # will print the value of z which is 15
print(‘The sum of x+y ‘, x+y)
print(“The sum of x+y :”, z)
print(‘Hawk’ + 30)              # will return an error message. You can not add string to number
print(‘Hawk ’ + ’30’)           # will print ‘Hawk 30’

print(‘\n’)                     # will enter an empty line followed by the default ‘\n’
print('\”Hello PLD’)            # will print ‘"Hello PLD’ – “ as a character
print(‘\’Hello PLD')            # will print ‘’Hello PLD’ – ‘ as a character
print('\tHello PLD’)            # will print ‘       Hello PLD’ – a Tab is inserted
print(‘Hello\\PLD’)             # will print ‘Hello\PLD’ – \ as a character
print(‘Hello\nHawk’)            # will print ‘Hello’ – newline – ‘Hawk’ on the newline
print(r’Hello\nHawk’)           # will print ‘Hello\nHawk’ – the r (raw string) cancels the role of the \

‘’’

‘’’
quantity = 3                    # argument number 0
itemno = 567                    # argument number 1
price = 49.95                   # argument number 2
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))


age = 40                        # argument number 0
txt = "My name is Babu, and I am {}"      # you can skip the argument number since it is only 1 argument
print(txt.format(age))
‘’’
```

❖ Programming Exercises

1. Personal Information

Design a Python program that displays the following information:

- Your name
- Your address, village/town/city, region
- Your telephone number
- Your email address
- Your college major

2. Sales Prediction

A company has determined that its annual profit is typically 23 percent of total sales. Design a Python program that asks the user to enter the projected amount of total sales, and then displays the profit that will be made from that amount.
Hint: Use the value 0.23 to represent 23 percent.

3. Land Calculation

One hectare of land is equivalent to 10,000 square meters. Design a Python program that asks the user to enter the total square meters in a tract of land and calculates the number of hectares in the tract.
Hint: Divide the amount entered by 10,000 to get the number of hectares.

4. Total Purchase

A customer in a store is purchasing five items. Design a Python program that asks for the price of each item, and then displays the subtotal of the sale, the amount of sales tax, and the total. Assume the sales tax is 6 percent.

5. Distance Traveled

Assuming there are no accidents or delays, the distance that a car travels down the interstate can be calculated with the following formula:

$$\textit{Distance} = \textit{Speed} \times \textit{Time}$$

A car is traveling at 60 miles per hour. Design a Python program that displays the following:

- The distance the car will travel in 5 hours
- The distance the car will travel in 8 hours
- The distance the car will travel in 12 hours

6. Sales Tax

Design a Python program that will ask the user to enter the amount of a purchase. The program should then compute the VAT tax. The program should display the amount of the purchase, the VAT tax, and the total of the sale (which is the sum of the amount of purchase plus the VAT tax).
Hint: Use the value 0.15 to represent 15%.

7. Area of a Circle

Design a Python program that will ask the user to enter the radius of a circle. The program should then compute the area of the circle and display the result. (*Hint: area of the circle is pi*r^2*)

8. Area of a Trapezoid

Design a Python program that will ask the user to enter the two bases of a trapezoid and height. The program should computer the area and display the result. (*Hint: area of the trapezoid is ½ *(base1 + base2)*height*)

9. Swap Two Numbers

Design a Python program that asks the user to enter two numbers. The program should swap the numbers and display.

10. Celsius to Fahrenheit Temperature Converter

Design a Python program that converts Celsius temperatures to Fahrenheit temperatures.
The formula is as follows:

$$F = (9/5)C + 32$$

The program should ask the user to enter a temperature in Celsius, and then display the temperature converted to Fahrenheit.

11. Stock Transaction Program

Last month Joe purchased some stock in Acme Software, Inc. Here are the details of the purchase:

- The number of shares that Joe purchased was 1,000.
- When Joe purchased the stock, he paid $32.87 per share.
- Joe paid his stockbroker a commission that amounted to 2 percent of the amount he paid for the stock.

Two weeks later Joe sold the stock. Here are the details of the sale:

- The number of shares that Joe sold was 1,000.
- He sold the stock for $33.92 per share.
- He paid his stockbroker another commission that amounted to 2 percent of the amount he received for the stock.

Design a Python program that displays the following information:

- The amount of money Joe paid for the stock.
- The amount of commission Joe paid his broker when he bought the stock.
- The amount that Joe sold the stock for.
- The amount of commission Joe paid his broker when he sold the stock.
- Did Joe make money or lose money? Display the amount of profit or loss after Joe sold the stock and paid his broker (both times).

❖ if … elif … else

- A decision structure allows a program to perform actions only under certain conditions
- Python supports:

Less than:	a **<** b
Less than or equal to:	a **<=** b
Greater than:	a **>** b
Greater than or equal to:	a **>=** b
Equal to:	a **==** b
Not Equal to:	a **!=** b

- Use **and** if you have two conditions that need to be true
- Use **or** if you have two conditions and one of them need to be true
- Use **not** to reverse the result of the condition

- Use the if statement to specify a block of code to be executed if a condition is **true**

```
'''
if condition:
    # statement(s) to be executed if the condition is true
'''

'''
a = 200
b = 33
if b > a:
    print("b is greater than a")

a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")

if a > b or a > c:
    print("At least one of the conditions is True")

if not b > a:
  print("b is NOT greater than a")
'''
```

- Use the if … else statement to execute statement(s) if the condition is **true** and another statement(s) if the condition is **false**

```
‘’’
if condition:
   # statement(s) to be executed if the condition is true
else:
   # statement(s) to be executed if the condition is false
‘’’
```

```
‘’’
a = 200
b = 33
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
‘’’
```

- Use if … elif … else statement to specify a new condition if the first condition is **false**

```
‘’’
if condition1:
    # statement(s) to be executed if condition1 is true

elif condition2:
    # statement(s) to be executed if the condition1 is false and condition2 is true
else:
    # statement(s) to be executed if the condition1 is false and condition2 is false
‘’’
```

```
‘’’
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
‘’’
```

- Use match-case instead if … elif … else when the result of condition is specific

```
'''
match expression:
    case pattern1:
        # statement(s) for pattern1
     case pattern2:
        # statement(s) for pattern2
     ...
     case patternN:
        # statement(s) for patternN
   case _:
        # default statement(s) if none of the above is true
'''

'''
day=int(input('Enter day of the week 1-7'))

match day:
    case 1:
        print('Monday')
    case 2:
        print('Tuesday')
    case 3:
        print('Wednesday')
    case 4:
        print('Thursday')
    case 5:
        print('Friday')
    case 6:
        print('Saturday')
    case 7:
        print('Sunday')
    case _:
        print('Error')
'''
```

❖ Programming Exercises

1. Roman Numerals

Design a Python program that prompts the user to enter a number within the range of 1 through 10. The program should display the Roman numeral version of that number. If the number is outside the range of 1 through 10, the program should display an error message.

2. Areas of Rectangles

The area of a rectangle is the rectangle's length times its width. Design a Python program that asks for the length and width of two rectangles. The program should tell the user which rectangle has the greater area, or whether the areas are the same.

3. Check Two Integers – Part1

Design a Python program to check two given integers and return true if one of them is 30 or if their sum is 30.

4. Check Two Integers – Part2

Design a Python program to check a given integer and return true if it is within 10 of 100 or 10 of 200.

5. Check Three Integers

Design a Python program to check the largest number among three given integers.

6. Mass and Weight

Scientists measure an object's mass in kilograms and its weight in Newtons. If you know the amount of mass of an object, you can calculate its weight, in Newtons, with the following formula:

$$Weight = Mass \times 9.8$$

Design a Python program that asks the user to enter an object's mass, and then calculates its weight. If the object weighs more than 1,000 Newtons, display a message indicating that it is too heavy. If the object weighs less than 10 Newtons, display a message indicating that it is too light.

7. Magic Dates

The date June 10, 1960, is special because when it is written in the following format, the month times the day equals the year:

6/10/60

Design a Python program that asks the user to enter a month (in numeric form), a day, and a two-digit year. The program should then determine whether the month times the day equals the year. If so, it should display a message saying the date is magic. Otherwise, it should display a message saying the date is not magic.

8. Color Mixer

The colors red, blue, and yellow are known as the primary colors because they cannot be made by mixing other colors. When you mix two primary colors, you get a secondary color, as shown here:

- When you mix red and blue, you get purple.
- When you mix red and yellow, you get orange.
- When you mix blue and yellow, you get green.

Design a Python program that prompts the user to enter the names of two primary colors to mix. If the user enters anything other than "red," "blue," or "yellow," the program should display an error message. Otherwise, the program should display the name of the secondary color that results.

9. Book Club Points

Serendipity Booksellers has a book club that awards points to its customers based on the number of books purchased each month. The points are awarded as follows:

- If a customer purchases 0 books, he or she earns 0 points.
- If a customer purchases 1 book, he or she earns 5 points.
- If a customer purchases 2 books, he or she earns 15 points.
- If a customer purchases 3 books, he or she earns 30 points.
- If a customer purchases 4 or more books, he or she earns 60 points.

Design a Python program that asks the user to enter the number of books that he or she has purchased this month and displays the number of points awarded.

10. Software Sales

A software company sells a package that retails for $99. Quantity discounts are given according to the following table:

Quantity	Discount
10–19	20%
20–49	30%
50–99	40%
100 or more	50%

Design a Python program that asks the user to enter the number of packages purchased. The program should then display the amount of the discount (if any) and the total amount of the purchase after the discount.

11. Shipping Charges

The Fast Freight Shipping Company charges the following rates:

Weight of Package	Rate per Pound
2 pounds or less	$1.10
Over 2 pounds but not more than 6 pounds	$2.20
Over 6 pounds but not more than 10 pounds	$3.70
Over 10 pounds	$3.80

Design a Python program that asks the user to enter the weight of a package and then displays the shipping charges.

12. Body Mass Index Program Enhancement

The Body Mass Index (BMI) is often used to determine whether a person with a inactive lifestyle is overweight or underweight for his or her height. A person's BMI is calculated with the following formula:

$$BMI = Weight \times 703 / Height^{\wedge 2}$$

In the formula, weight is measured in pounds and height is measured in inches. Design a Python program that asks the user to enter the weight and height. The program should calculate the BMI and display a message indicating whether the person has optimal weight, is underweight, or is overweight. An inactive person's weight is considered to be optimal if his or her BMI is between 18.5 and 25. If the BMI is less than 18.5, the person is considered to be underweight. If the BMI value is greater than 25, the person is considered to be overweight.

13. Time Calculator

Design a Python program that asks the user to enter a number of seconds, and works as follows:

- There are 60 seconds in a minute. If the number of seconds entered by the user is greater than or equal to 60, the program should display the number of minutes in that many seconds.
- There are 3,600 seconds in an hour. If the number of seconds entered by the user is greater than or equal to 3,600, the program should display the number of hours in that many seconds.
- There are 86,400 seconds in a day. If the number of seconds entered by the user is greater than or equal to 86,400, the program should display the number of days in that many seconds.

❖ while Loop

- The while loop loops through a block of code as long as a specified condition is **true**
- **Initialize** outside the loop to make condition true
- Pre-Test the condition before executing the code
- Inside the loop, have an option to **change** the condition to false (prevents infinite loop)
- Use **break** statement to stop the loop even if the while condition is true
- Use **continue** statement to stop the **current** iteration, and continue with the next

```
'''
while condition:
    # statement(s) to be executed if the condition is true
'''

'''
x = 1                   # Initialize the value to make the condition true
while x < 6:            # Pre-test the condition
    print(x)
    x += 1              # Change the value allowing the condition to become false to exit the loop
'''

'''
x = 1
while x < 6:
    print(x)
    if x == 3:
        break
    x += 1
'''

'''
x = 0
while x < 6:
    x += 1
    if x == 3:
        continue
    print(x)
'''
```

❖ for Loop

- Use a for loop to iterate over sequences
- **break** and **continue** statements can be used
- Use **range(***start, end, step***)** to generate a sequence of numbers for the loop
  - If *step* is not given, a list of consecutive numbers will be generated with step **1**
  - The given *end* value is **never** part of the generated sequence of numbers – **end-1**
- You can loop through a string (other sequences will be discussed later)
- You can construct multi-level (**nested**) loops where the "inner loop" will be executed one time for each iteration of the "outer loop"

```
'''
for variable in range(start, end, step):
    # statement(s) to be executed
'''

'''
for x in range(1, 15, 1):
    print(x)

for x in range(2, 30, 3):
    print(x)
'''

'''
lang='Python'

for x in lang:
    print(x)                # will print the letters of the word Python each on a separate line


for i in 'good luck':       # will print the letters of the sentence each on a separate line – including the space
   print(i)
'''

'''
for x in range(5, 1, -1):
    print("Outer Loop: ",  x)
    for y in range(1, 4, 1):
        print(" Inner Loop: ", y)
'''
```

❖ Programming Exercises

1. Total of a Series of Numbers
Design a Python program that asks the user to enter a positive integer number and calculates the total of the following series of numbers: 1/n + 2/(n-1) + 3/(n-2) + 4/(n-3) + 5/(n-4) … + n/1
(Hint, if you enter 4 then the program will calculate the total of 1/4 + 2/3 + 3/2 + 4/1)

2. Calories Burned
Running on a particular treadmill you burn 3.9 calories per minute. Design a Python program that uses a loop to display the number of calories burned after 10, 15, 20, 25, and 30 minutes.

3. Sum of Numbers – Part1.
Design a Python program with a loop that asks the user to enter a series of positive numbers. The user should enter a negative number to signal the end of the series. After all the positive numbers have been entered, the program should display their sum.

4. Sum of Numbers – Part2.
Design a Python program that calculates and prints the sum of cubes of even numbers up to a specified limit (e.g., 20) using a while loop.

5. Sum of a Series of Integers – Part1.
Design a Python program to calculate the sum of the series (1*1) + (2*2) + (3*3) + (4*4) + (5*5) + ... + (n*n). The output should look like:

```
1*1 = 1
2*2 = 4
3*3 = 9
…
The sum of the above series is:
```

6. Sum of a Series of Integers – Part2.
Design a Python program to calculate the series (1) + (1+2) + (1+2+3) + (1+2+3+4) + ... + (1+2+3+4+...+n). The output should look like:

```
1 = 1
1+2 = 3
1+2+3 = 6
…
The sum of the above series is:
```

7. Right Angle Patter
Design a Python program that makes a pattern such as a right-angle triangle using numbers that repeat.

```
1
22
333
4444
55555
…
nnnnn
```

8. Tuition Increase
At one college, the tuition for a full-time student is $6,000 per semester. It has been announced that the tuition will increase by 2 percent each year for the next five years. Design a Python program with a loop that displays the projected semester tuition amount for the next five years.

9. Series of Integers
Design a Python program to read a series of integers. The first integer is special, as it indicates how many more integers will follow. The output of the program will be the calculated the sum and average of the integers [excluding the first integer].

10. Pennies for Pay
Design a Python program that calculates the amount of money a person would earn over a period of time if his or her salary is one penny the first day, two pennies the second day, and continues to double each day. The program should ask the user for the number of days. Display the total pay at the end of the period. The output should be displayed in a dollar amount, not the number of pennies.

11. Largest and Smallest
Design a Python program with a loop that lets the user enter a series of numbers. The user should enter –99 to signal the end of the series. After all the numbers have been entered, the program should display the largest and smallest numbers entered.

12. First and Last
Design a Python program that asks the user for a series of names (in no particular order). After the final person's name has been entered, the program should display the name that is first alphabetically and the name that is last alphabetically. For example, if the user enters the names Kristin, Joel, Adam, Beth, Zeb, and Chris, the program will display Adam and Zeb.

❖ String Manipulation

- Remember that string is a group of Unicode characters. Hence, each character can have an index starting from **0**

```
'''
a = "Hello, Babu!"              # a is a string variable of length 12 characters
print(a[1])                     # will print e
'''
```

- Use **len()** to find the length of a string

```
'''
a = "Hello, Babu!"
print(len(a))                   # will return 12 – the number of characters
'''
```

- Use the keyword **in** to check if a certain phrase or character is present in a string. Also, use not in to check if a certain phrase or character is NOT present in a string

```
'''
txt = "The best things in life are free!"
if "free" in txt:
    print("Yes, 'free' is present.")
else:
    print("No, 'free is not present.")

if "expensive" not in txt:
    print("Yes, 'expensive' is NOT present.")
'''
```

- You can get a slice of string by specifying where to *start* and *end*. Remember since indexing starts from zero, the end number is **not included**

```
'''
a = "Hello, Babu!"

x=a[2:5]
print(x)                # will print 'llo'

y=a[:5]                 # if you do not include the start, Python will assume zero
print(y)                # will print 'hello'

z=a[2:]                 # if you do not include the end, Python will assume till the end of string
print(z)                # will print 'llo, babu!'
'''
```

- Use **upper()** to return the string in UPPER CASE
  Use **lower()** to return the string in lower case
  Use **swapcase()** to return the string where all the upper case letters are lower case and vice versa
  Use **capitalize()** to return the string where the first character is upper case, and the rest is lower case
  Use **title()** to return the string where the first character in every word is upper case. If the word contains a number or symbol, the first letter after that will be converted to upper case

```
‘’’
x = "Hello, Babu!"
print(x.upper())                # will print ‘HELLO, BABU!’
print(x.lower())                # will print 'hello, babu!’
print(x.swapcase())             # will print ' hELLO, bABU!’
print(x.capitalize())           # will print ' Hello, babu!’

y=” Welcome to my world”
print(y.title())                # will print 'Welcome To My World’

z=”Welcome to my 2nd world”
print(z)                        # will print ‘Welcome To My 2Nd World’
‘’’
```

- Use **strip()** to remove any **<u>whitespace</u>** from the beginning or the end

```
‘’’
a='  Hello, Babu!  '
print(a.strip())                # will print ‘Hello, Babu!’ – all whitespaces before and after were removed
‘’’
```

- Use **replace()** to replace part of a string with a character or string

```
‘’’
a=’Hello, Babu!’
x=a.replace(‘He’, ‘je’)         # x=’jello, Babu!’
print(x)
‘’’
```

- Use **center(***length*, *character***)** to center a string surrounded by the characters within the length specified

```
‘’’
x = "Hello, Babu!"
print(x.center(20, ‘*’))        # will print ‘****Hello, Babu!****’
‘’’
```

- Use **split()** to split the string into substrings. The split happens based on the character(s)

```
‘’’
x = "Hello, Babu!"
print(x.split(','))               # will print 'Hello', ' Babu!'
print(x.split('e'))               # will print 'H', 'llo, Babu!'
print(x.split(‘ ‘)                # will print ‘Hello,’ , ‘Babu!’
‘’’
```

- Use **count()** to return the number of times a specified value appears in a string

```
‘’’
x=’Hello, Babu! Welcome to PLD class’
print(x.count('Welcome'))         # will print 1
print(x.count('c'))               # will print 2
print(x.count(' '))               # will print 5
‘’’
```

- Use find() to find the first occurrence of the specified value. If not found, the returned value is **-1**

```
‘’’
a= "Hello, welcome to my world."
print(a.find('welcome'))          # will print 7
print(a.find('come'))             # will print 10
print(a.find('Welcome'))          # will print -1
‘’’
```

❖ Programming Exercises

1. **Count digits in a string**

Design a Python program that will ask the user to enter a string. The program will count the sum of all digits in the string.

2. **Reverse a string**

Design a Python program that will ask the user to enter a string. The program will reverse and display the string.

3. **Remove non alphabet characters**

Design a Python program that will ask the user to enter a string. The program will remove all non-alphabet characters (digits and symbols) and display the result.

4. **Create a mix string**

Design a Python program that will ask the user to enter three strings. The program will check the length of each string and use the length of the shortest as reference. Then the program will create and display a fourth string with the first character from each string, followed by the second character from each string till reaching the reference length.

❖ List

- A collection of data which are normally related
- Individual values in the list are accessible by their **<u>indexes</u>**, and indexes always start from **zero** to **size-1**
- You can access the values of a list from the back. The last item in the list has an index of **-1**, the second last has an index of **-2** and so forth
- You can:
  assign and slice to a variable – note: when using x = alist, x will be a reference to alist
  append an element to the end of the list / insert an element at a specific index
  extend elements from another list to the current list
  join two or more lists
  copy a list (it is not the same as assigning a list to a variable.
    copy() will just copy the elements of the list. This means that any changes to the new variable will not be reflected to the original list
  reverse the elements of the list / sort the elements of a list ascending / descending
  remove specific element / delete element with specific index
  clear the list
  for / **while** loop through the list

```
'''
listName = [value1, value2, value3, …, valueN]

#or

listName=[]
'''

'''
userAge = [21, 22, 23, 24, 25]

print(userAge [1])          # will print 22
print(userAge [-1])         # will print 25

x = userAge                 # assign a list to a variable
print(x)

y = userAge[2:4]            # assign the 2nd & 3rd elements to y – remember that the end is not included
print(y)                    # will print [23, 24]

userAge.append(99)          # will append 99 to the end of the list
print(userAge)              # will print [21, 22, 23, 24, 25. 99]

userAge.insert(2, 33)       # insert at index 2 the 33
print(userAge)              # will print [21, 22, 33, 23, 24, 25, 99]

v=[60, 70, 80, 90]
userAge.extend[v]           # extend userAge by adding the lements of v at the end
print(userAge)              # will print [21, 22, 23, 24, 25, 60, 70, 80, 90]
```

```
userAge.remove(90)            # remove element 90
print(userAge)                # will print [21, 22, 23, 24, 25, 60, 70, 80]

del userAge[3]                # delete element with index 3
print(userAge)                # will print [21, 22, 23, 25, 60, 70, 80]
k=userAge

print(k)                      # will print [21, 22, 23, 25, 60, 70, 80]
k.clear()                     # clear k
print(k)                      # will print []


aList=[22, 23, 24, 25]

for i in aList:
    print(i)

for i in range(len(aList)):
    print(aList[i])

i = 0
while i < len(aList):
    print(aList[i])
    i = i + 1


fruits = ["orange", "mango", "kiwi", "pineapple", "banana"]
x=fruits
x.sort()                      # sort x ascending
print(x)                      # will print ['banana', 'kiwi', 'mango', 'orange', 'pineapple']

y=fruits
y.sort(reserse=True)          # sort y descending
print(y)                      # will print ['pineapple', 'orange', 'mango', 'kiwi', 'banana']

z=fruits
z.reverse()                   # reverse elements in list
print(z)                      # will print ['banana', 'pineapple', 'kiwi', 'mango', 'orange']

k=x.copy()                    # copy elements of x, but x not assigned to k
print(k)                      # will print ['banana', 'kiwi', 'mango', 'orange', 'pineapple']


x = ["a", "b" , "c"]
y = [1, 2, 3, 4]
z = x + y                     # join x and y
print(z)                      # will print ['a', 'b', 'c', 1, 2, 3, 4]
‘’’
```

❖ Programming Exercises

1. Difference between max and min

Design a Python program that asks the user to create a list of numbers. The program should ask the user when to stop inserting numbers in the list. Then, the program should find the difference between the largest and smallest numbers in the list.

2. Common elements in two lists

Design a Python program that asks the user to create two lists of numbers of any sizes . Then, the program should find the common elements between the two lists.

3. Sum and product of numbers

Design a Python program that asks the user to create a list of numbers. Then, the program should find the sum and product of numbers with odd indices.

4. Convert a list of characters into a string

Design a Python program that asks the user to create a list of characters. Then, the program should convert it

5. Interchange numbers

Design a Python program that ask the user to create an even number list of digits. Then, the program should swap the first digit with the last digit, 2nd digit with the before last digit …etc.

❖ Tuples

- Tuples are just like lists, but you **cannot** modify their values. The initial values are the values that will stay for the rest of the program
- Python allows to extract the values back into variables – unpacking a tuple

```
'''
tupleName = (value1, value2, value3, …, valueN)
'''

'''
monthsOfYear = ("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")

print(monthsOfYear[3])              # will print 'Apr'
print(monthsOfYear[-2])             # will print 'Nov'


myTuple = (60, 70, 80, 90)
x, y, z, w = myTuple                # unpacking a tuple into variables

print(x)                            # will print 60
…
print(w)                            # will print 90
'''

'''
work around to be able to change a tuple

-   convert the tuple to list
-   change the created list
-   convert the list to a tuple

mytuple = (60, 70, 80, 90)
y = list(mytuple)
y[0] = 100                          # change 60 to 100
y.append(987)                       # add 987 to end of list
mytuple = tuple(y)
print(mytuple)                      # will print (100, 70, 80, 90, 987)
'''
```

❖ Dictionary

- Dictionary is a collection of related data pairs – key and data
- You can change, add or remove items after the dictionary has been created
- Within a dictionary, dictionary keys must be unique – no duplicates
- Dictionary keys and data can be of different data types
- Dictionaries are used in data storage. Programmers do not need to remember which index refers to which data. Instead, the keys can refer to data
- You can use
  len() to determine how many items a dictionary
  keys() to return a list of all the keys in the dictionary
  values() to return a list of all the values in the dictionary
  items() to return a list of each item of data with its key
  in to check if a key exists or not in a dictionary
  del / pop() to remove an item with a specific key
  for to loop through a dictionary
  copy() to create a non-referenced copy of a dictionary

```
‘’’
dictionaryName = { }                   # can be used to populate the dictionary with data & keys

# or

dictionaryName = {dictionary key : data}
‘’’

‘’’
carDict = {
           "brand": "Ford",
           "model": "Mustang",
           "year": 1964
          }

print(carDict)                         # will print {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
print(carDict[‘model’])                # will print ‘Mustang’
print(carDict.keys())                  # will print dict_keys(['brand', 'model', 'year'])
print(carDict.values())                # will print dict_values(['Ford', 'Mustang', 1964])
print(carDict.items())                 # will print dict_items([('brand', 'Ford'), ('model', ‘Mustang’), ('year', 1964)])

carDict[‘color’] = ‘White’             # add a new element to dictionary
print(carDict)                         # will print {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'White'}

# or

carDict[‘color’]) = [‘Red’, ‘White’, ‘Blue’] # add a new element with list of data
print(carDict)                         # will print {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': ['Red',
                                       'White', 'Blue']}
```

```
carDict[‘year’] = 2010                    # change the data from 1964 to 2010
print(carDict['year’])                    # will print 2010

if ‘model’ in carDict:                    # check if key exists or not
    print(‘Yes. ‘model’ is a key in carDict’)

carDict.pop(‘color’)                      # remove data with key ‘color’

# or

del carDict[‘color’]                      # remove data with key ‘color’

print(carDict)                            # will print {'brand': 'Ford', 'model': 'Mustang', 'year': 2010}



for x in carDict:                         # will print keys
    print(x)

# or

for x in carDict.keys():
    print(x)


for x in carDict:                         # will print values
    print(thisdict[x])

# or

for x in thisdict.values():
    print(x)


for x, y in carDict.items():              # will print key and value of each element in the dictionary
  print(x, y)


myCar = carDict.copy()                    # copy a non-referenced copy to myCar
print(myCar)                              # will print {'brand': 'Ford', 'model': 'Mustang', 'year': 2010}
‘’’
```

❖ Random

- You need to include at the beginning of code: **import** random
- import is similar to #include in C++. It is used to access modules in the Python libraries
- You can generate random integer / float numbers
- You need to include the module name that imported
- Use
  random.random() to generate a float number between **0** & **1**
  random.randint(*min*, *max***)** to generate an integer number between **min** & **max**
  random.sample(*range(min, max)*, *numbersinlist*) to generate a list of random numbers – number in list – between **min** & **max**
  random.choice(*myList***)** to select a random number from a list

```
'''
import random

x = random.random()
print(x)                          # will print a random float number between 0 !

x = random.randint(1, 10)
print(x)                          # will print a random integer number between 1 & 10

x = random.sample(range(1, 100), 15)
print(x)                          # will print a list of 15 randomly generated numbers between 1 & 100


myList = [1,3,5,7,9]
x = random.choice(myList)
print(x)                          # will print a selected random number from the list
'''
```

❖ Programming Exercises

1. Rock, Paper, Scissors Game

Design a Python program that lets the user play the game of Rock, Paper, Scissors against
the computer. The program should work as follows:

i. When the program begins, a random number in the range of 1 through 3 is generated. If the number is 1, then the computer has chosen rock. If the number is 2, then the computer has chosen paper. If the number is 3, then the computer has chosen scissors. (Don't display the computer's choice yet).
ii. The user enters his or her choice of "rock," "paper," or "scissors" at the keyboard.
iii. The computer's choice is displayed.
iv. The program should display a message indicating whether the user or the computer was the winner.

A winner is selected according to the following rules:

- If one player chooses rock and the other player chooses scissors, then rock wins. (The rock smashes the scissors).
- If one player chooses scissors and the other player chooses paper, then scissors wins. (Scissors cut paper).
- If one player chooses paper and the other player chooses rock, then paper wins. (Paper wraps rock).
- If both players make the same choice, the game must be played again to determine the winner.

2. Lottery Number Generator

Design a Python program that will generate a 7-digit lottery number in a list. The randomly generated numbers are from 0 to 49.

3. Roll the dice

Design a Python program that rolls three dice. The program should ask the user if wants to continue after a roll.

- If you roll three sixes, print: 'best hand, triple 6'
- If you roll three of a kind, print: 'pretty good, triple x', replace x with number
- If you roll a sequence, print: 'nice'
- For any other result, print 'too bad'

4. Slot Machine Simulation

A slot machine is a gambling device that the user inserts money into and then pulls a lever (or presses a button). The slot machine then displays a set of random images. If two or more of the images match, the user wins an amount of money, which the slot machine dispenses back to the user.
Design a Python program that simulates a slot machine. When the program runs, it should do the following:

- Ask the user to enter the amount of money he or she wants to insert into the slot
machine.
- Instead of displaying images, the program will randomly select a word from the following list:

Cherries, Oranges, Plums, Bells, Melons, Bars

The program will select and display a word from this list three times.

- If none of the randomly selected words match, the program will inform the user that he or she has won $0. If two of the words match, the program will inform the user that he or she has won two times the amount entered. If three of the words match, the program will inform the user that he or she has won three times the amount entered.

❖ Functions / Modules

- A function is a block of code which only runs when it is **called**
- You can pass data to a function – parameter(s) that can be any type
- Use default parameter value – if the function is called without a parameter, it will use the default value
- A function can return data as a result
- A function is defined above the code that calls it
- Use **def** to define a function
- To call a function use the function name followed by parenthesis – **()**
- When using modular programming, variables are classified into **global** and **local** variables

```
'''
def function_name(parameter(s))
    # statement(s) to be executed


function_name(parameter(s))              # call a function
'''

'''
def name_func(fname, lname):
  print(fname + " " + lname)


name_func("Sweet", "Babu")               # call the function
'''

'''
def my_func(country = "Gambia"):
  print("Hello from " + country)


my_func("Sweden")
my_func("USA")
my_func()                                # default value will be used
my_func("Brazil")
'''

'''
def my_func(x):
    y = 5 * x
    return y


print(my_func(5))
print(my_func(8))
'''
```

❖ Programming Exercises

1. Kilometer Converter

Design a Python modular program that asks the user to enter a distance in kilometers, and then converts that distance to miles. The conversion formula is as follows:

Miles = Kilometers × 0.6214

2. Property Tax

A county collects property taxes on the assessment value of property, which is 60 percent of the property's actual value. For example, if an acre of land is valued at $10,000, its assessment value is $6,000. The property tax is then 64¢ for each $100 of the assessment value. The tax for the acre assessed at $6,000 will be $38.40. Design a Python modular program that asks for the actual value of a piece of property and displays the assessment value and property tax.

3. Calories from Fat and Carbohydrates

A nutritionist who works for a fitness club helps members by evaluating their diets. As part of her evaluation, she asks members for the number of fat grams and carbohydrate grams that they consumed in a day. Then, she calculates the number of calories that result from the fat, using the following formula:

Calories from Fat = Fat Grams × 9

Next, she calculates the number of calories that result from the carbohydrates,
using the following formula:

Calories from Carbs = Carb Grams × 4

The nutritionist asks you to design a Python modular program that will make these calculations.

4. Stadium Seating

There are three seating categories at a stadium. For a football game, Class A seats cost $15, Class B seats cost $12, and Class C seats cost $9. Design a Python modular program that asks how many tickets for each class of seats were sold, and then displays the amount of income generated from ticket sales.

5. Paint Job Estimator

A painting company has determined that for every 115 square feet of wall space, one gallon of paint and eight hours of labor will be required. The company charges $20.00 per hour for labor. Design a Python modular program that asks the user to enter the square feet of wall space to be painted and the price of the paint per gallon. The program should display the following data:

- The number of gallons of paint required
- The hours of labor required
- The cost of the paint
- The labor charges
- The total cost of the paint job

6. Feet to Inches

One foot equals 12 inches. Design a Python program with function named ***feetToInches*** that accepts a
number of feet as a parameter and returns the number of inches in that many feet. Use the function in a program that prompts the user to enter a number of feet and then displays the number of inches in that many feet.

7. Falling Distance

When an object is falling because of gravity, the following formula can be used to determine the distance the object falls in a specific time period:

$$d = 1/2gt^{\wedge 2}$$

The variables in the formula are as follows: d is the distance in meters, g is 9.8, and t is the amount of time, in seconds, that the object has been falling. Design a Python function named ***fallingDistance*** that accepts an object's falling time (in seconds) as an argument. The function should return the distance, in meters, that the object has fallen during that time interval. Design a Python program that calls the function in a loop that passes the values 5 through 10 as parameters and displays the return value.

8. Kinetic Energy

In physics, an object that is in motion is said to have kinetic energy. The following formula can be used to determine a moving object's kinetic energy:

$$ke = 1/2mv^{\wedge 2}$$

The variables in the formula are as follows: KE is the kinetic energy, m is the object's mass in kilograms, and v is the object's velocity, in meters per second. Design a Python function named ***kineticEnergy*** that accepts an object's mass (in kilograms) and velocity (in meters per second) as parameters. The function should return the amount of kinetic energy that the object has. Design a Python program that asks the user to enter values for mass and velocity, and then calls the kineticEnergy function to get the object's kinetic energy.

9. Test Average and Grade

Design a Python program that asks the user to enter five test scores. The program should display a letter grade for each score and the average test score. Design the following functions in the program:

- ***calcAverage*** – This function should accept five test scores as parameters and return the average of the scores.
- ***determineGrade*** – This function should accept a test score as a parameter and return a letter grade for the score (as a String), based on the following grading scale:

Score	Letter Grade
90–100	A
80–89	B
70–79	C
60–69	D
Below 60	F

❖ Built in Functions

- Use **abs(*number*)** to return the absolute value of a number

```
‘’’
a = 12
b = -4
c = 7.90

print(abs(a))                 # will print 12
print(abs(b))                 # will print 4
print(abs(c))                 # will print 7.9
‘’’
```

- Use **bin(*number*)** to return the binary value of a number

```
‘’’
x = 9

print(bin(9))                 # will print ob1001
‘’’
```

- Use **list(*object*)** to convert and return a list – an object can be string or tuple

```
‘’’
x = “Hello”
print(list(x))                # will print ['h', 'e', 'l', 'l', 'o']

y = (‘a’, ‘e’, ‘i’, ‘o’, ‘u’)
print(list(y))                # will print ['a', 'e', 'i', 'o', 'u']
‘’’
```

- Use **bool(*object*)** to return the Boolean value of an abject. The function will return False for zero or None or empty string

```
‘’’
print(bool(0))                # will print False
print(bool(-4.5))             # will print True
print(bool(None))             # will print False
print(bool("False"))          # will print True – it is a string with the word False
print(bool(‘ ‘))              # will print True – it is a string with space character
print(bool(‘’))               # will print False – it is an empty string
‘’’
```

- Use **round(*number*)** to return the nearest integer to a number

```
'''
print(round(4.6))              # will print 5
print(round(4.5))              # will print 4
print(round(6.2))              # will print 6
print(round(-7.7))             # will print -8
print(round(-4.3))             # will print -4
'''
```

- Use **min(*object*)** / **max(*object*)** to return the minimum / maximum of a list / tuple / string or a group of characters / number

```
'''
print(max('hello', 'Babu'))    # will print 'hello'
print(max(9, 6, 15))           # will print 15
print(min(2, 5, 3, 1, 0, 99))  # will print 0
print(min(['B','a','t','A']))  # will print 'A'
'''
```

- Use **pow(*number*, *x*)** to return the number to the power of x – number^x

```
'''
print(pow(5, 4))               # will print 625
print(pow(2, -3))              # will print 0.125
print(pow(2, 4.5))             # will print 22.627416997969522
print(pow(3, 0))               # will print 1
'''
```

- Use **sum(*object*)** to return the sum of all numbers in a list / tuple. Also, you can sum of the object to a number

```
'''
num = [2.5, 3, 4, -5]
print(sum(num))                # will print 4.5
print(sum(num, 25))            # will print 29.5
'''
```

- Use **chr(*ascii*)** to return the character that represents the specified unicode

```
'''
print(chr(97))                 # will print 'a'
print(chr(68))                 # will print 'D'
'''
```

❖ Math Functions

- You need to include at the beginning of code: **import** math
- import is similar to #include in C++. It is used to access modules in the Python libraries
- You need to include the module name that imported

- Use **sqrt(***number***)** to return the square root of a number

```
'''
import math

x = math.sqrt(64)
print(x)
'''
```

- Use pi to return the value of pi

```
'''
import math

x = math.pi
print(x)
'''
```

- Use **ceil(***number***)** / **floor(***number***)** to round a number **upwards** / **downwards** to its nearest integer

```
'''
import math

x = math.ceil(1.4)
y = math.floor(1.4)

print(x)                    # will print 2
print(y)                    # will print 1
'''
```

- Use **degrees(***angleRadian***)** / **radians(***angleDegrees***)** to convert the angle to **degrees** / **radians** – PI radians = 180 degrees

```
'''
import math

print(math.radians(30))
print(math.degrees(3.14))
'''
```

- Use **cos(***angleRadian***)**, **sin(***angleRadian***)**, **tan(***angleRadian***)** to return cos / sin / tan of an angle – angle is in radians

```
‘’’
import math

print(math.sin(math.radians(30)))          # will print the sign of 30 degrees
print(math.cos(math.radians(45)))          # will print the cos of 45 degrees
‘’’
```

- Use **factorial(***number***)** to return the factorial of a number

```
‘’’
import math

print(math.factorial(3)                    # will print 6 – 3! = 3*2*1
‘’’
```

- Use **prod(***object***)** to return the multiplication of elements in a list / tuple

```
‘’’
import math

print(math.prod((2,3,4,5)))                # will print 120
‘’’
```

❖ Using Lists instead of Arrays

- To use a list instead of array, you need to make sure that all the elements of the list is of the **same data type**
- A list can be utilized as a 1D array
- A list of lists can be utilized as a 2D array – each element has two indices

```
'''
A list of lists – 2D

variable = [ [elements of 1st list],          # the 1st list has index 0
             [elements of 2nd list],          # the 2nd list has index 1
             …
             [elements of Nth list] ]         # the Nth list has index N-1
'''

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

print(x[0][1])                                # will print 2
print(x[2][2])                                # will print 9

y = x[1][2] + x[0][1]
print(y)                                      # will print 8
'''
```

❖ Programming Exercises

1. **Total Sales**

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

2. **Rainfall Statistics**

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

3. **Number Analysis Program**

Design a C++ program that asks the user to enter a series of 20 numbers. The program should store the numbers in a list and then display the following data:

- The lowest number
- The highest number
- The total of the numbers
- The average of the numbers

4. **Charge Account Validation**

Design a Python program that asks the user to enter a charge account number. The program should determine whether the number is valid by comparing it to the following list of valid charge account numbers:

5658845	4520125	7895122	8777541	8451277	1302850
8080152	4562555	5552012	5050552	7825877	1250255
1005231	6545231	3852085	7576651	7881200	4581002

These numbers should be stored in a tuple. If the number is in the tuple, the program should display a message indicating the number is valid. If the number is not in the tuple, the program should display a message indicating the number is invalid.

5. **Payroll**

Design a Python program that uses the following **parallel** list:

- ***empId***: A list of seven Integers to hold employee identification numbers. The list should be initialized with the following numbers: 56588 45201 78951 87775 84512 13028 75804
- ***hours***: A list of seven Integers to hold the number of hours worked by each employee.
- ***payRate***: A list of seven Reals to hold each employee's hourly pay rate.
- ***wages***: A list of seven Reals to hold each employee's gross wages.

The program should relate the data in each list through the subscripts. For example, the number in element 0 of the hours list should be the number of hours worked by the employee whose identification number is stored in element 0 of the empId list. That same employee's pay rate should be stored in element 0 of the payRate list.
The program should display each employee number and ask the user to enter that employee's hours and pay rate. It should then calculate the gross wages for that employee (hours times pay rate), which should be stored in the wages list. After the data has been entered for all the employees, the program should display each employee's identification number and gross wages.
(*Can you try this problem using a dictionary?!!!*)

6. Vowels

Design a Python program that will randomly generate 50 alphabet letters into a list. After that, the program will output the randomly generated alphabet letters followed by the number of vowels A, E, I, O, U on the next line.
Hint: The ASCII values for A-Z are 65-90

7. Driver's License Exam

The local driver's license office has asked you to design a Python program that grades the written portion of the driver's license exam. The exam has 20 multiple choice questions. Here are the correct answers:

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

Your program should store these correct answers in a tuple. The program should ask the user to enter the student's answers for each of the 20 questions, which should be stored in a list. After the student's answers have been entered, the program should display a message indicating whether the student passed or failed the exam. (A student must correctly answer 15 of the 20 questions to pass the exam.) It should then display the total number of correctly answered questions, and the total number of incorrectly answered questions.

8. Tic-Tac-Toe Game

Design a Python program that allows two players to play a game of tic-tac-toe. Use a 2D string list of lists with three rows and three columns as the game board. Each element of the list should be initialized with an asterisk (*). The program should run a loop that does the following:
a. Displays the contents of the board list.
b. Allows player 1 to select a location on the board for an X. The program should ask the user to enter the row and column number.
c. Allows player 2 to select a location on the board for an O. The program should ask the user to enter the row and column number.
d. Determines whether a player has won or if a tie has occurred. If a player has won, the program should declare that player the winner and end. If a tie has occurred, the program should say so and end.
e. Player 1 wins when there are three Xs in a row on the game board. Player 2 wins when there are three Os in a row on the game board. The winning Xs or Os can appear in a row, in a column, or diagonally across the board. A tie occurs when all of the locations on the board are full, but there is no winner.

❖ Files

- Python has several functions for creating, reading, updating, and deleting files
- Start with **Open(‘*fileName*’, *mode*)** – assigned to a variable
- **filename** parameter is a string that includes where the file is located and name
- **mode** parameter specifies if you are going to read, write, or append data to the file
  - **‘r’** – Read – Default value. Opens a file for reading, error if the file does not exist
  - **‘a’** – Append – Opens a file for appending, creates the file if it does not exist
  - **‘w’** – Write – Opens a file for writing, creates the file if it does not exist
- You can not read a file that is opened to append or write
- End with variable.**close()** to close the file

- Use **read()** to read the contents of the file

```
‘’’
xFile = open (‘fileName’, ‘r’)
print(xFile.read())                 # will print the contents of the file
print(xFile.read(7))                # will print the first 7 characters that are read
xFile.close()
‘’’
```

- Use **readline()** to read a whole line till **endofline**

```
‘’’
xFile = open (‘fileName’, ‘r’)
print(xFile.readline())             # will print the first line followed by ‘\n’
print(xFile.readline(), end = ‘’)   # will print the first line NOT followed by ‘\n’
print(xFile.readline())             # will print the second line followed by '\n’
xFile.close()
‘’’
```

- You can loop through a file to read it line by line

```
‘’’
xFile = open (‘fileName’, ‘r’)

for x in xFile:
    print(x)
xFile.close()
‘’’
```

- Use **write()** to write to a file

```
‘’’
xFile = open (‘fileName’, ‘a’)      # open a file to add text
xFile.write(‘text added’)           # text will be added after the last character in the file
xFile.close()
‘’’
```

```
‘’’
xFile = open (‘fileName’, ‘w’)          # open a file to write text. If the file already exists and has text, writing will wipe
                                        out the text and replace it with the written one
xFile.write(‘text written’)
xFile.close()
‘’’
```

❖ Programming Exercises

1. Series of Numbers in a File
Assume that a file containing a series of integers is named **numbers.dat** and exists on the computer's disk. Design a Python program that reads all the numbers and displays:

- A list of the numbers
- The sum of the numbers
- The average of the numbers
- The largest and smallest numbers

2. Rows of Series of Numbers in a File
Design a Python program that asks the user to enter two positive integer numbers. The first integer is the number of rows of numbers that will be saved in a file named **rowsData.txt** on the computer's disk. The second integer is the number of numbers in each row. After that, the program will open the file and read the data to display:

- The sum of number is each row
- The total of all numbers entered
- The average of all numbers entered
- The largest and smallest numbers

3. PLD Students Data in a File
Design a Python program that asks the user to enter the data of each student in the PLD class in a list of dictionaries. A dictionary consists of:

- MAT
- FirstName
- LastName
- Midterm
- Final

The data of the list will be saved in a file named PLDStudents.txt on the computer's disk. After that, the program will open the file and read the data to display:

- MAT FirstName LastName Midterm Final TotalGrade #of all students
- The average of all TotalGRade