Practice Sheet 1 - Python Programming

Asst. Prof. Syed Faisal Ali Programming Fundamentals - FALL 2019 Software Engineering
Dated: 7 Nov 2019

Q1: Show how we can import 10 differnt functions from math library of python, such as trignometric or hyperbolic functions and two constants pi and e, and store 6 different angles such as zero, thirty, forty five, sixty, ninty and one hundred and eighty. Show them how you will use in program by giving them some values and print them.

In [28]:
```
from math import sin,cos,tan,factorial,exp,sqrt,radians,degrees
from math import pi,e

print ("pi = ",pi)
print ("e = ",e)

a1 = 0
a2 = 30
a3 = 45
a4 = 60
a5 = 90
a6 = 180

#SINE
print("\nSine:")
a1_sin = sin(a1)
a2_sin = sin(a2)
a3_sin = sin(a3)
a4_sin = sin(a4)
a5_sin = sin(a5)
a6_sin = sin(a6)
print(a1_sin)
print(a2_sin)
print(a3_sin)
print(a4_sin)
print(a5_sin)
print(a6_sin)

#COSINE
print("\nCosine:")
a1_cos = cos(a1)
a2_cos = cos(a2)
a3_cos = cos(a3)
a4_cos = cos(a4)
a5_cos = cos(a5)
a6_cos = cos(a6)
print(a1_cos)
print(a2_cos)
print(a3_cos)
print(a4_cos)
print(a5_cos)
print(a6_cos)

#TANGENT
print("\nTangent:")
a1_tan = tan(a1)
a2_tan = tan(a2)
a3_tan = tan(a3)
a4_tan = tan(a4)
a5_tan = tan(a5)
a6_tan = tan(a6)
print(a1_tan)
print(a2_tan)
print(a3_tan)
print(a4_tan)
print(a5_tan)
```

```
print(a6_tan)

#RADIANS
print("\nRadians:")
a1_rad = radians(a1)
a2_rad = radians(a2)
a3_rad = radians(a3)
a4_rad = radians(a4)
a5_rad = radians(a5)
a6_rad = radians(a6)
print(a1_rad)
print(a2_rad)
print(a3_rad)
print(a4_rad)
print(a5_rad)
print(a6_rad)

A = 1
B = 2
C = 3
D = 4
E = 5
F = 6

#SQUARE ROOT
print("\nSqaure Root:")
sqrt1 = sqrt(A)
sqrt2 = sqrt(B)
sqrt3 = sqrt(C)
sqrt4 = sqrt(D)
sqrt5 = sqrt(E)
sqrt6 = sqrt(F)
print(sqrt1)
print(sqrt2)
print(sqrt3)
print(sqrt4)
print(sqrt5)
print(sqrt6)
```

```
pi =  3.141592653589793
e =  2.718281828459045

Sine:
0.0
-0.9880316240928618
0.8509035245341184
-0.3048106211022167
0.8939966636005579
-0.8011526357338304

Cosine:
1.0
0.15425144988758405
0.5253219888177297
-0.9524129804151563
-0.4480736161291701
-0.5984600690578581

Tangent:
0.0
-6.405331196646276
1.6197751905438615
0.320040389379563
-1.995200412208242
1.3386902103511544

Radians:
0.0
0.5235987755982988
0.7853981633974483
1.0471975511965976
1.5707963267948966
3.141592653589793

Sqaure Root:
1.0
1.4142135623730951
1.7320508075688772
2.0
2.23606797749979
2.449489742783178
```

Q2. Use help() command to show what is math library, int, and class.

In [1]:
```
help('math')

help(int)

help('class')
```

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.

    copysign(x, y, /)
        Return a float with the magnitude (absolute value) of x but the sign
of y.

        On platforms that support signed zeros, copysign(1.0, -0.0)
        returns -1.0.

    cos(x, /)
        Return the cosine of x (measured in radians).

    cosh(x, /)
        Return the hyperbolic cosine of x.

    degrees(x, /)
        Convert angle x from radians to degrees.

    erf(x, /)
        Error function at x.
```

```
    erfc(x, /)
        Complementary error function at x.

    exp(x, /)
        Return e raised to the power of x.

    expm1(x, /)
        Return exp(x)-1.

        This function avoids the loss of precision involved in the direct eva
luation of exp(x)-1 for small x.

    fabs(x, /)
        Return the absolute value of the float x.

    factorial(x, /)
        Find x!.

        Raise a ValueError if x is negative or non-integral.

    floor(x, /)
        Return the floor of x as an Integral.

        This is the largest integer <= x.

    fmod(x, y, /)
        Return fmod(x, y), according to platform C.

        x % y may differ.

    frexp(x, /)
        Return the mantissa and exponent of x, as pair (m, e).

        m is a float and e is an int, such that x = m * 2.**e.
        If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

    fsum(seq, /)
        Return an accurate floating point sum of values in the iterable seq.

        Assumes IEEE-754 floating point arithmetic.

    gamma(x, /)
        Gamma function at x.

    gcd(x, y, /)
        greatest common divisor of x and y

    hypot(x, y, /)
        Return the Euclidean distance, sqrt(x*x + y*y).

    isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)
        Determine whether two floating point numbers are close in value.

          rel_tol
            maximum difference for being considered "close", relative to the
            magnitude of the input values
          abs_tol
```

```
            maximum difference for being considered "close", regardless of th
e
            magnitude of the input values

        Return True if a is close in value to b, and False otherwise.

        For the values to be considered close, the difference between them
        must be smaller than at least one of the tolerances.

        -inf, inf and NaN behave similarly to the IEEE 754 Standard.  That
        is, NaN is not close to anything, even itself.  inf and -inf are
        only close to themselves.

    isfinite(x, /)
        Return True if x is neither an infinity nor a NaN, and False otherwis
e.

    isinf(x, /)
        Return True if x is a positive or negative infinity, and False otherw
ise.

    isnan(x, /)
        Return True if x is a NaN (not a number), and False otherwise.

    ldexp(x, i, /)
        Return x * (2**i).

        This is essentially the inverse of frexp().

    lgamma(x, /)
        Natural logarithm of absolute value of Gamma function at x.

    log(...)
        log(x, [base=math.e])
        Return the logarithm of x to the given base.

        If the base not specified, returns the natural logarithm (base e) of
x.

    log10(x, /)
        Return the base 10 logarithm of x.

    log1p(x, /)
        Return the natural logarithm of 1+x (base e).

        The result is computed in a way which is accurate for x near zero.

    log2(x, /)
        Return the base 2 logarithm of x.

    modf(x, /)
        Return the fractional and integer parts of x.

        Both results carry the sign of x and are floats.

    pow(x, y, /)
        Return x**y (x to the power of y).
```

```
    radians(x, /)
        Convert angle x from degrees to radians.

    remainder(x, y, /)
        Difference between x and the closest integer multiple of y.

        Return x - n*y where n*y is the closest integer multiple of y.
        In the case where x is exactly halfway between two multiples of
        y, the nearest even value of n is used. The result is always exact.

    sin(x, /)
        Return the sine of x (measured in radians).

    sinh(x, /)
        Return the hyperbolic sine of x.

    sqrt(x, /)
        Return the square root of x.

    tan(x, /)
        Return the tangent of x (measured in radians).

    tanh(x, /)
        Return the hyperbolic tangent of x.

    trunc(x, /)
        Truncates the Real x to the nearest Integral toward 0.

        Uses the __trunc__ magic method.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    (built-in)


Help on class int in module builtins:

class int(object)
 |  int([x]) -> integer
 |  int(x, base=10) -> integer
 |
 |  Convert a number or string to an integer, or return 0 if no arguments
 |  are given.  If x is a number, return x.__int__().  For floating point
 |  numbers, this truncates towards zero.
 |
 |  If x is not a number or if base is given, then x must be a string,
 |  bytes, or bytearray instance representing an integer literal in the
 |  given base.  The literal can be preceded by '+' or '-' and be surrounded
 |  by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
 |  Base 0 means to interpret the base from the string as an integer literal.
```

```
|  >>> int('0b100', base=0)
|  4
|
|  Methods defined here:
|
|  __abs__(self, /)
|      abs(self)
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __and__(self, value, /)
|      Return self&value.
|
|  __bool__(self, /)
|      self != 0
|
|  __ceil__(...)
|      Ceiling of an Integral returns itself.
|
|  __divmod__(self, value, /)
|      Return divmod(self, value).
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __float__(self, /)
|      float(self)
|
|  __floor__(...)
|      Flooring an Integral returns itself.
|
|  __floordiv__(self, value, /)
|      Return self//value.
|
|  __format__(self, format_spec, /)
|      Default object formatter.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __index__(self, /)
|      Return self converted to an integer, if self is suitable for use as a
n index into a list.
|
|  __int__(self, /)
```

```
|      int(self)
|
|  __invert__(self, /)
|      ~self
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __lshift__(self, value, /)
|      Return self<<value.
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mod__(self, value, /)
|      Return self%value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __neg__(self, /)
|      -self
|
|  __or__(self, value, /)
|      Return self|value.
|
|  __pos__(self, /)
|      +self
|
|  __pow__(self, value, mod=None, /)
|      Return pow(self, value, mod).
|
|  __radd__(self, value, /)
|      Return value+self.
|
|  __rand__(self, value, /)
|      Return value&self.
|
|  __rdivmod__(self, value, /)
|      Return divmod(value, self).
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __rfloordiv__(self, value, /)
|      Return value//self.
|
|  __rlshift__(self, value, /)
|      Return value<<self.
|
|  __rmod__(self, value, /)
|      Return value%self.
|
|  __rmul__(self, value, /)
```

```
|      Return value*self.
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __round__(...)
|      Rounding an Integral returns itself.
|      Rounding with an ndigits argument also returns an integer.
|
|  __rpow__(self, value, mod=None, /)
|      Return pow(value, self, mod).
|
|  __rrshift__(self, value, /)
|      Return value>>self.
|
|  __rshift__(self, value, /)
|      Return self>>value.
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rtruediv__(self, value, /)
|      Return value/self.
|
|  __rxor__(self, value, /)
|      Return value^self.
|
|  __sizeof__(self, /)
|      Returns size in memory, in bytes.
|
|  __str__(self, /)
|      Return str(self).
|
|  __sub__(self, value, /)
|      Return self-value.
|
|  __truediv__(self, value, /)
|      Return self/value.
|
|  __trunc__(...)
|      Truncating an Integral returns itself.
|
|  __xor__(self, value, /)
|      Return self^value.
|
|  bit_length(self, /)
|      Number of bits necessary to represent self in binary.
|
|      >>> bin(37)
|      '0b100101'
|      >>> (37).bit_length()
|      6
|
|  conjugate(...)
|      Returns self, the complex conjugate of any int.
|
|  to_bytes(self, /, length, byteorder, *, signed=False)
```

```
 |      Return an array of bytes representing an integer.
 |
 |      length
 |        Length of bytes object to use.  An OverflowError is raised if the
 |        integer is not representable with the given number of bytes.
 |      byteorder
 |        The byte order used to represent the integer.  If byteorder is 'bi
g',
 |        the most significant byte is at the beginning of the byte array.  I
f
 |        byteorder is 'little', the most significant byte is at the end of t
he
 |        byte array.  To request the native byte order of the host system, u
se
 |        `sys.byteorder' as the byte order value.
 |      signed
 |        Determines whether two's complement is used to represent the intege
r.
 |        If signed is False and a negative integer is given, an OverflowErro
r
 |        is raised.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  from_bytes(bytes, byteorder, *, signed=False) from builtins.type
 |      Return the integer represented by the given array of bytes.
 |
 |      bytes
 |        Holds the array of bytes to convert.  The argument must either
 |        support the buffer protocol or be an iterable object producing byte
s.
 |        Bytes and bytearray are examples of built-in objects that support t
he
 |        buffer protocol.
 |      byteorder
 |        The byte order used to represent the integer.  If byteorder is 'bi
g',
 |        the most significant byte is at the beginning of the byte array.  I
f
 |        byteorder is 'little', the most significant byte is at the end of t
he
 |        byte array.  To request the native byte order of the host system, u
se
 |        `sys.byteorder' as the byte order value.
 |      signed
 |        Indicates whether two's complement is used to represent the intege
r.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signatur
e.
 |
 |  ----------------------------------------------------------------------
```

```
 |  Data descriptors defined here:
 |
 |  denominator
 |      the denominator of a rational number in lowest terms
 |
 |  imag
 |      the imaginary part of a complex number
 |
 |  numerator
 |      the numerator of a rational number in lowest terms
 |
 |  real
 |      the real part of a complex number
```

Class definitions
****

A class definition defines a class object (see section The standard type hierarchy):

```
   classdef    ::= [decorators] "class" classname [inheritance] ":" suite
   inheritance ::= "(" [argument_list] ")"
   classname   ::= identifier
```

A class definition is an executable statement. The inheritance list usually gives a list of base classes (see Metaclasses for more advanced uses), so each item in the list should evaluate to a class object which allows subclassing. Classes without an inheritance list inherit, by default, from the base class "object"; hence,

```
   class Foo:
       pass
```

is equivalent to

```
   class Foo(object):
       pass
```

The class’s suite is then executed in a new execution frame (see Naming and binding), using a newly created local namespace and the original global namespace. (Usually, the suite contains mostly function definitions.) When the class’s suite finishes execution, its execution frame is discarded but its local namespace is saved. [3] A class object is then created using the inheritance list for the base classes and the saved local namespace for the attribute dictionary. The class name is bound to this class object in the original local namespace.

The order in which attributes are defined in the class body is preserved in the new class’s "__dict__". Note that this is reliable only right after the class is created and only for classes that were defined using the definition syntax.

Class creation can be customized heavily using metaclasses.

Classes can also be decorated: just like when decorating functions,

```
   @f1(arg)
   @f2
   class Foo: pass

is roughly equivalent to

   class Foo: pass
   Foo = f1(arg)(f2(Foo))

The evaluation rules for the decorator expressions are the same as for
function decorators.  The result is then bound to the class name.

**Programmer's note:** Variables defined in the class definition are
class attributes; they are shared by instances.  Instance attributes
can be set in a method with "self.name = value".  Both class and
instance attributes are accessible through the notation ""self.name"",
and an instance attribute hides a class attribute with the same name
when accessed in this way.  Class attributes can be used as defaults
for instance attributes, but using mutable values there can lead to
unexpected results.  Descriptors can be used to create instance
variables with different implementation details.

See also:

  **PEP 3115** - Metaclasses in Python 3000
     The proposal that changed the declaration of metaclasses to the
     current syntax, and the semantics for how classes with
     metaclasses are constructed.

  **PEP 3129** - Class Decorators
     The proposal that added class decorators.  Function and method
     decorators were introduced in **PEP 318**.

Related help topics: CLASSES, SPECIALMETHODS
```

Q3. Use len() function to show the lenght of your name stored in a variable my_name.

In [1]:
```
my_name=('Muhammad Shoaib')
b =(len(my_name)-2)
#As it had to spaces in between
print (b)
```

```
13
```

Q4: We use type() function to show the type of the variable. Your task is to use it on different type of variables in which you save your name, date of birth, cell number, email, favourite dish, favourite color, favourite movie, favourite flower, etc.

In [6]:
```
name = ('Muhammad Shoaib')
a = type(name)
print (a)

dob = 17-9-1999
b = type(dob)
print (b)

phone_no = +92351080450
c = type(phone_no)
print (c)

email = ('siddiquiumer200@gmail.com')
d = type(email)
print (d)

fav_dish = ('pasta')
e = type(fav_dish)
print (e)

fav_color = ('Black')
f = type(email)
print (f)

fav_movie = ("martian")
g = type(fav_movie)
print (g)

fav_flower = ('rose')
h = type(fav_flower)
print(h)
```

```
<class 'str'>
<class 'int'>
<class 'int'>
<class 'str'>
<class 'str'>
<class 'str'>
<class 'str'>
<class 'str'>
```

Q5: A football is kicked 1 meter from the ground at an angle of 30 degree with a velocity of 15 m/sec. Calculate the following:

a. Max height above the ground.

b. Total time in the air.

c. Total distance traveled.

d. Velocity vector at max height.

e. Velocity when the ball hits the ground.

In [40]:
```
from math import sin

v = 15
print("Velocity is : 15 m/sec")
g = (-9.8)
ang = 30
i =1

#FOR MAX HEIGHT:
h = (v * sin(ang)/g)+1
print ("Max height of projectile is "+ str(h)+ " m from the ground.")

#FOR TOTAL TIME IN AIR:
t = 2*v*sin(2*ang)/g
print ("Total time in air is : "+str(t)+" seconds")

#FOR TOTAL DISTANCE TRAVELLED:
d = v**2 * (sin(2*ang))/g
print("Total distance travelled is :"+str(d)+" m")

#FOR VELOCITY ALONG X-AXIS:
Vx = v* i
print ("Velocity vector at max height :" +str(Vx)+ " m")

#FOR FINAL VELOCITY
f_vel = h*(-g)
print ("Final velocity :"+str(f_vel)+" m/sec")
```

```
Velocity is : 15 m/sec
Max height of projectile is 2.5122933021829517 m from the ground.
Total time in air is : 0.9330937380680102 seconds
Total distance travelled is :6.998203035510077 m
Velocity vector at max height :15 m
Final velocity :24.62047436139293 m/sec
```

Q6: A mass of 5 kg is placed in an inclined plane. If 10 N force applied on it and the friction on the inclined plane is 0.4. Find, Sin, Cos and its movement for angle 30, 45 and 60 degrees ?

 mass

In [2]:
```
from math import sin,cos

mass = 5
grav = 9.8
force = 10
fric = 0.4

ang1 = 30
ang2 = 45
ang3 = 60

#FOR 30°:
a = grav*sin(ang1)*(-fric/mass)
print("Acceleration : "+str(a)+" m/s²")
f1 = mass * acc +fric-sin(30)*mass*grav
print(f1)

#FOR 45°:
b = -(grav*sin(ang2)*(-fric/mass))
print("Acceleration : "+str(b)+" m/s²")
f2 = mass * acc +fric-sin(45)*mass*grav
print(f2)

#FOR 60°:
c = grav*sin(ang3)*(-fric/mass)
print("Acceleration : "+str(c)+" m/s²")
f3 = mass * c +fric-sin(60)*mass*grav
print(f3)
```

```
Acceleration : 0.7746167932888038 m/s²
52.68663354699425
Acceleration : 0.667108363234749 m/s²
-37.42118873572779
Acceleration : 0.23897152694413792 m/s²
16.530578068729305
```

Q7: Three resiters of 10, 20 and 30 ohms are connected in series and in parallel if 10 V is given to them find current and voltages across them.

In [27]:
```
r1 = 10
r2 = 20
r3 = 30
v = 10
print ("GIVEN:\n\tFirst Resistor = 10 Ohms \n\tSecond Resistor = 20 Ohms \n\tT
hird resistor = 30 Ohms \n\tVolts = 10 volts ")

series = (r1 + r2 + r3)
print ("\nEquivalent resistance of these resistors in series is: ",series," OH
MS")

parallel = (1/(1/r1 + 1/r2 + 1/r3))
print ("\nEquivalent resistance of these resistors in parallel is: ",parallel,
"OHMS\n")

I_s = v/series
print ("Current in series : "+str(I_s)+" amp\n")

#VOLTAGE ACROSS EACH RESISTOR IN SERIES
v_s1 = (I_s * r1)
v_s2 = (I_s * r2)
v_s3 = (I_s * r3)
print (v_s1,"volts")
print (v_s2,"volts")
print (v_s3,"volts")
print("")
#CURRENT ACROSS EACH RESISTOR IN PARALLEL
i1 = v/r1
i2 = v/r2
i3 = v/r3
print(i1,"amp")
print(i2,"amp")
print(i3,"amp")
```

```
GIVEN:
        First Resistor = 10 Ohms
        Second Resistor = 20 Ohms
        Third resistor = 30 Ohms
        Volts = 10 volts

Equivalent resistance of these resistors in series is:  60  OHMS

Equivalent resistance of these resistors in parallel is:  5.454545454545454 O
HMS

Current in series : 0.16666666666666666 amp

1.6666666666666665 volts
3.333333333333333 volts
5.0 volts

1.0 amp
0.5 amp
0.3333333333333333 amp
```

Q8. Find the point of intercept if line 1 coordinates are (1,5), (7, 9) and line 2 coordinates are (2,4) (6,9).

In [3]:
```
#CO-ORDINATES OF POINT "A"
x1 = 1
y1 = 5

#CO-ORDINATES OF POINT "B"
x2 = 7
y2 = 9

#CO-ORDINATES OF POINT "C"
x3 = 2
y3 = 4

#CO-ORDINATES OF POINT "D"
x4 = 6
y4 = 9

slope1 = (y2 - y1)/(x2 -x1)
slope2 = (y4 - y3)/(x4 -x3)


print(slope1)
print(slope2)
```

```
0.6666666666666666
1.25
```

Q9: Write a small program which have unit price of 10 items such as eggs Rs 120 dozen, milk Rs 95 liter, butter 100 Rs for 125 gms, etc Now ask from the user one by one what he wants and how many he wants. Then do the total amount of purchase.

In [14]:
```
print("***Welcome to ABC Store***")

lst = ("\nEggs = 120Rs per dozen \nMilk = 95Rs per litre \nButter = 100Rs per
 125 grms \nBread = 100Rs per loaf \nMeat = 450Rs per KG \nTissue = 55Rs per b
ox \nShampoo = 175Rs per 500ml \nSoap = 45Rs per bar \npen = 15Rs per unit  \n
Pencil = 5Rs per unit \nNote book = 75Rs per unit")
print (lst)
print ("")

a = float(input("How many dozens of eggs do you want : "))
a1 = (a * 120)

b = float(input("How many litres of milk do you want : "))
b1 = (b * 95)

c = int(input("How many 125gm packs of butter do you want : "))
c1 = (c * 100)

d = int(input("How many loaves of bread do you want : "))
d1 = (d * 100)

e = int(input("How many KGs of meat do you want : "))
e1 = (e * 450)

f = int(input("How many boxes of tissue do you want : "))
f1 = (f * 55)

g = int(input("How many 500ml shampoo bottles do you want : "))
g1 = (g * 175)

h = int(input("How many bars of soap do you want : "))
h1 = (h * 45)

i = int(input("How many pens do you want : "))
i1 = (i * 15)

j = int(input("How many pencils do you want : "))
j1 = (j * 5)

k = int(input("How many notebooks do you want : "))
k1 = (k * 75)

tot = (a1 + b1 + c1 + d1 + e1 + f1 + g1 + h1 + i1 + j1 + k1)
print ("\nKindly pay ",tot,"Rs for your items.")

print("\n\nThanks for your visit\n***Good bye**" )
```

```
***Welcome to ABC Store***

Eggs = 120Rs per dozen
Milk = 95Rs per litre
Butter = 100Rs per 125 grms
Bread = 100Rs per loaf
Meat = 450Rs per KG
Tissue = 55Rs per box
Shampoo = 175Rs per 500ml
Soap = 45Rs per bar
pen = 15Rs per unit
Pencil = 5Rs per unit
Note book = 75Rs per unit

How many dozens of eggs do you want : 1.5
How many litres of milk do you want : 2.5
How many 125gm packs of butter do you want : 1
How many loaves of bread do you want : 2
How many KGs of meat do you want : 3
How many boxes of tissue do you want : 5
How many 500ml shampoo bottles do you want : 4
How many bars of soap do you want : 6
How many pens do you want : 8
How many pencils do you want : 9
How many notebooks do you want : 7

Kindly pay  4002.5 Rs for your items.


Thanks for your visit
***Good bye**
```

Q10: A 5m 60cm high vertical pole casts a shadow 3m 20 cm long. Find at the same time
(i) the length of the shadow cast another pole 10m 50cm high
(ii) the height of a pole which casts a shadow 5m long.

In [42]:
```
#FRIST FIND THE RATIO OF HEIGHT OF POLE AND SHADOW'S LENGHT :
a = 560/320

#THEN USWE IT TO FIND RESPECTIVE ANSWERS
b = (1050/a)/100
print("length of shadow is : ",b)

c = (a*500)/100
print("Height of pole is : ",c)
```

```
length of shadow is :  6.0
Height of pole is :  8.75
```

In [ ]: