Algorithm - finite sequence of (unambiguous instructions; solving a problem; for any legitimate input; in a finite amount of time[should finish at some point])
Computers are very good at executing [very basic instructions; very fast; on very large amounts of data]

type(23) = int
type(23.0) = float
type("adc") = str
If type is float result is float.

Base10
0,1,2,..9
$3678_{10} = 3\times10^3 + 5\times10^2 + 7\times10^1 + 8\times10^0$

Base 8
0,1,2 ... 7
$13618 = 1\times8^3 + 7\times8^1 + 6\times8^0$
0,1,2,... 7, 10,11,12, ... 17,20

Base 16
0,1,2,... 9,A,B,C,D,E,F
$1A2_{16} \rightarrow 1\times16^2 + A\times16^1 + 2\times16^0 = 418_{10}$
0,1,2 ... 9,A,B ... F,10,11 .... 19,1A ... 1F,20

Base 2
0, 1
$1110_2 = 1\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0 = 14_{10}$
0, 1, 10, 11, 100, 101, 110, 111, 1000

Binary System Conversion
- 2 → 10
  $1101_{(2} \rightarrow 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$
  $= 13_{(10}$
- 10 → 2

2 | 13	1101
2 | 6	1
2 | 3	0
2 | 1	1
0	1

Variable - name of an area of memory containing some value. Container holding a value.
Types: Integers; Booleans; Strings; float
Input = numbers; raw_input = letters and symbols.
Name of constants: all uppercase.

Fixed-point: Signed binary
- Use the leftmost bit[sign bit] to indicate the sign: 0 = positive; 1 = negative.

2's compliment:
0001 [-1 in 1's compliment] -> flip it
1000 -> add 1
1111 = -1 in 2's compliment

Memory :- storing both bata and instructions.
Internal representation of numbers in Python functions:
- ord("A") = decimal code for "A" in ASCII.
- hex(ord("A")) = hexadecimal code for "A"
- bin(ord("A")) = binary code for "A"

Signed w/ 2's Compliment
- 0000 to 1111
- 7 positive 0 8 negative
  $-8 \le x \le 7$
  $-2^{n-1} \le x \le 2^{n-1} - 1$

Complier or interpreters translate code written in a language into machine language instructions.
n bits - 2^n [2bits=2^3; 3bit=2^8]

Function len ():
len("abc") = 3

Indexing a string:
nameN = "Peyton"
print nameN[3]
>> t

Slicing a string:
nameN = "Peyton"
print nameN[1:4]
>>> eyt

Get first number in variable.
num = 512.3
numS = str(num)
print numS[0]

Boolean Expression:- an expression that is evaluated a Boolean value: True or False.
not, and, or

Relational Operators: > < => <= == !=
Built-in: varS.isdigit(), varS.isalpha(), bool()

Truth Table

A	B	A and B	A or B
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T

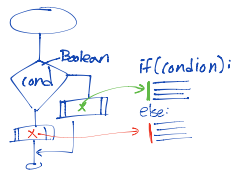
Repeated Concatenation:
"ha" * 3 = hahaha; "3" * 5 = 33333
Basic Operation: () **[exponentiation] * / % + - (PEMDAS) (work left to right)

Functions: Built-in [already provided by Python]
Created by us [def my function())]

Fruitful [RETURN with the statement; return some value or object; "real mathematical function"]
Void [no value is returned, but something is done (ex. print)]

Operations and methods:
round(num[,n]) - value num rounded to n digits after the decimal point
import math - [math.pi, math.sqrt(4) = 2.0]
import random
- random.randint(a,b): returns a random number N between a and b
- random.random(): return a random floating point number in [0.0, 1.0]

Flow chart Symbols
- → start/stop
- → one statement
- → input/output statement
- → condition
- → sequence of statement
- → for loop

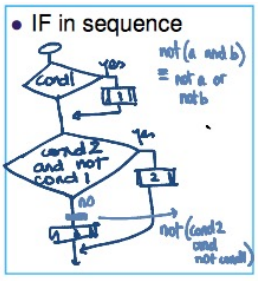


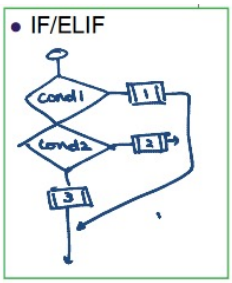

```
def Name ():
    first_last_names = name.split()
    return first_last_names [0][0] + first_last_names [1][0]
def Address ():
    fletteradd = address.find(" ") + 1
    f_ad = address [fletteradd]
    if (f_ad.isdigit()):
        return ""
    else:
        return f_ad
def Sentence ():
    letters = sentence.split()
    return letters[0][0] + letters [1][0] + letters[2][0]
n1 = len(s1) + math.pi + ord(s1[0])

def Code():
    if (code[0:1].isalpha() == False or code[1:2].isalpha()==False):
        return code [0:1]
    else:
        return code

def messagepartg():
    if (len(abcdef)%2 == 0):
        return "E"
    else:
        return "O"

exclamluck = "!!"*(luck())
abc = name[1].upper() + str(luck ()) + exclamluck
abcd = name[1].upper() + str(luck ()) + exclamluck +
str(address.find(name[-1]))
abcde = name[1].upper() + str(luck ()) + exclamluck +
str(address.find(name[-1])) + str(Code())
abcdef = abcde + s1
finalmessage = abcdef + messagepartg()

print "\nContructing the message..."

print "\nTRACE so far, parts a,b,c, the message is...", abc
print "TRACE so far, parts a,b,c,d the message is...", abcd
print "TRACE so far, parts a,b,c,d,e the message is...", abcde

print "\n *** Yey! the message is ", finalmessage
```

```
yy_dif = total_dif_days/D_YEAR
left_over_days = total_dif_days % D_YEAR
mm_dif = left_over_days /D_MONTH

dd_dif = left_over_days % D_MONTH
```

```
exclam = "!!"
find_comma = oddr.find(",") - 1
l = oddr[find_comma]
print l

oddrlist = oddr.split()
postal_last = oddrlist[-1][-1]
dollarsign = int(postal_last) * "$"
print postal_last
print dollarsign

firstpostal = oddrlist[-1][0]
print firstpostal

n = oddr [0:2]
import random
addr5 = random.randint(1,int(n))

if (addr5%2 == 0):
    print exclam + l + dollarsign + firstpostal + str(addr5) + "-an odd message"
else:
    print exclam + l + dollarsign + firstpostal + str(addr5)
```

DECIMAL (base 10)	BINARY (base 2)	HEXADECIMAL (base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F





import turtle as t
t.forward(dist), t.backward(dist) – distance
t.right(deg), t.left(deg) - degrees
t.penup(), t.pendown()
t.width(), t.width(w) - width pen ex: t.width(3)
t.home(), t.goto(x,y), t.clear()
t.colormode(255): To be able to use colors with 3 #'s in (1,255)
t.pencolor("red"), t.pencolor((n1,n2,n3))
t.fillcolor(<color>), t.begin_fill(), t.end_fill()

```
if (<condition>):
    ## (1) statements
    ##      when cond is T)
else:
    ## (2) statements
    ##     when cond is F)
```

```
If (a):
   if (b):
       ## (1) when
       ##      a and b  are T)
    else:
       ## (2) when  a T,  b F
else:
   ## (3) when a F
   ##     (b may be T or F)
```

```
if (<condition - 1>):
        ## (1) cond-1 T
elif (<condition-2>):
        ## (2) cond-1 F,
        ##     cond-2 T
...
elif (condition –n>):
        ## (n) cond-1 F ,
        ##      cond-2 F,
        ##      cond-n T
else:
        ##  (*) all cond's F
```

```
for   <var>  in  <sequence>:
      ------------------------
Ex:   for  k  in range(5) :
          print k*100 + 2
Ex:  for  ch  in "abcd"
          print ch + "!":
```

Repeated accumulation

```
acc= 0
for k in ......
        # could include if
        acc = acc + .... (k)....
```

Repeated string construction

```
newst = ""
for k in ......
        # could include if
        newst  = newst  + .... (k)....
```

What does each version print?

All versions start with
ns = ""
name = "abcde"

A
```
for  k in range(len(name)):
        ns = name[k]
print ns
```
'e'

B
```
for k in range(len(name)):
        ns + name[k]
print ns
```
original value of ns

C
```
for  k in range(len(name)):
        ns = ""
        ns = ns + name[k]
print ns
```
'e'

D
```
for  k in range(len(name)):
        ns = ns + name[k]
print ns
```
'abcde'

```
def add_digits_in_string(st):
    res = 0
    for ch in st:
        if ch.isdigit():
            res = res + int(ch)
    return res

def digits_plus (n):
    res = ""
    for i in range(n+ 1):
        res = res + str(i) + "+"
    return res
```

```
def maxDigs(st):
max = -1
    for i in range(len(st)):
        if st[i].isdigit():
            digI = int(st[i])
            if digI > max:
                max = digI
    return max

def immPairs(st):
    count = 0
    for i in range(1,len(st)):
        if st[i-1] == st[i]:
            count = count + 1
```

```
def cookies (n): huge = n/48
    left_after_huge = n %48
    small = left_after_huge /8
    unused = left_after_huge %8
    money = huge *26 + small * 4 -unused*2
    res = str(huge) + ".." + str(small) + ".." + str(unused) + ".." + str(money)
    return res

def repeat_middle(st):
    if (len(st)%2 == 0):
        mid_pos = len(st)/2 -1
        mid_char = st[mid_pos]+st[mid_pos+1] res = mid_char*len(st)
        res = "!!" + res + "!!"
    else:
        mid_pos = len(st)/2
        mid_char = st[mid_pos] res = mid_char*len(st)
        res = "!" + res + "!"
    return res

def count_digits(st):
    count = 0
    for i in range(len(st)):
        if st[i].isdigit():
            count = count + 1
return count


def countLower(st):
    count = 0
    i=0
    while i < len(st):
```

```
def inBetween(st):
    res = ""
    i=0
    while (i < len(st)):
        res = res + st[i] + "*"
        i=i+1
    return res

def inBetweenAdv(st):
    res = ""
    for i in range(len(st)):
        res = res + st[i]
        if st[i].isdigit():
            res = res + "D"
        else:
            res = res + "X"
    res = res[0:len(res)-1]
    return res
```

```
            if st[i].isalpha():
                if st[i].islower():
                    count = count + 1
            i=i+1
    return count


def countLower(st,start):
    count = 0
    i = start
    while i < len(st) and not(st[i].isdigit()):
            if st[i].isalpha():
                if st[i].islower():
                    count = count + 1
            i=i+1
    return count

def avgBackw(st):
    sum = 0.0
    count = 0
    i = len(st) - 1
    while (i >=0 and not(st[i].isalpha())):
            if st[i].isdigit():
                sum = sum + int(st[i])
                count = count + 1
            i=i–1
    if count == 0:
            res = 0.0
    else:
            res = sum/count
    return res


Flag 1. Print for every:
for i in range (num):
    if glasses (i):
            print True
    else:
            print False

Flag 2. Only last one
for i in range (num):
    if glasses (i):
            found = True
    else:
            found = False
print found

Flag 3. Once. Flag Change
found = False
for i in range (num):
    if glasses (i):
            found = True
print found


Example Flag
def difActionFlag (st,ch):
    found = False
    total = 0
    for i in range (len(st)):
            if st[i] == ch:
                found = True
            if found:
                total = total + 2
            else:
                total = total + 1
    return total
```