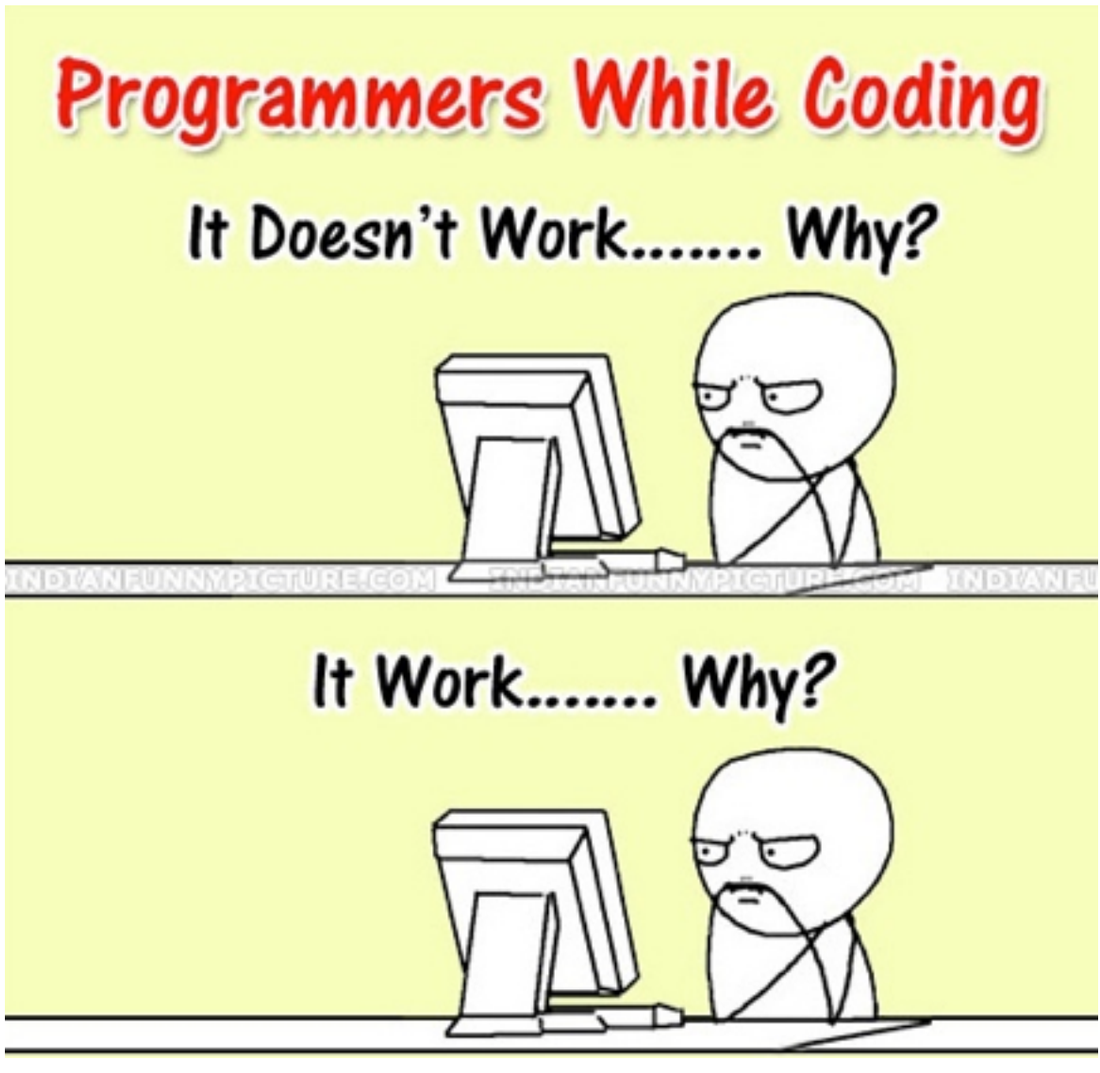
WELCOME!

Introduction to Computation

IDC 101

Course Credit: 3



Source: Google

Course Instructors:

Bijay Agarwalla
(Physics)
Course Coordinator

M S Santhanam
(Physics)

Umakant Rapol
(Physics)

Sarvesh Dube
(Earth Science)

Lectures and Labs:

Lectures (11-13):

Friday (11:30 am - 12:30 am)

Labs :

Monday Tuesday, Thursday, Friday

(2 pm-4 pm)

Source: Google

Class will be divided into 4 batches

Friday: Batch 1 (20181001-20181050)
Monday: Batch 2 (20181051-20181100)
Tuesday: Batch 3 (20181101-20181150)
Thursday: Batch 4- all others,

Office Hours: Friday (5pm - 6pm)

Office No: A 385

Evaluation

1. **Quiz**- 20% – (Beginning of September)
2. **Mid-sem**- 30% -2 hrs (24th Sept-1st October)
3. **Quiz** -20% -(Beginning of November)
4. **End-sem**- 30% -2 hrs (22nd Nov-30th Nov)

Assignments: After every lecture (no weightage)

Course Material: Website
http://www.iiserpune.ac.in/~santh/idc101_2018.html

will be updated every week

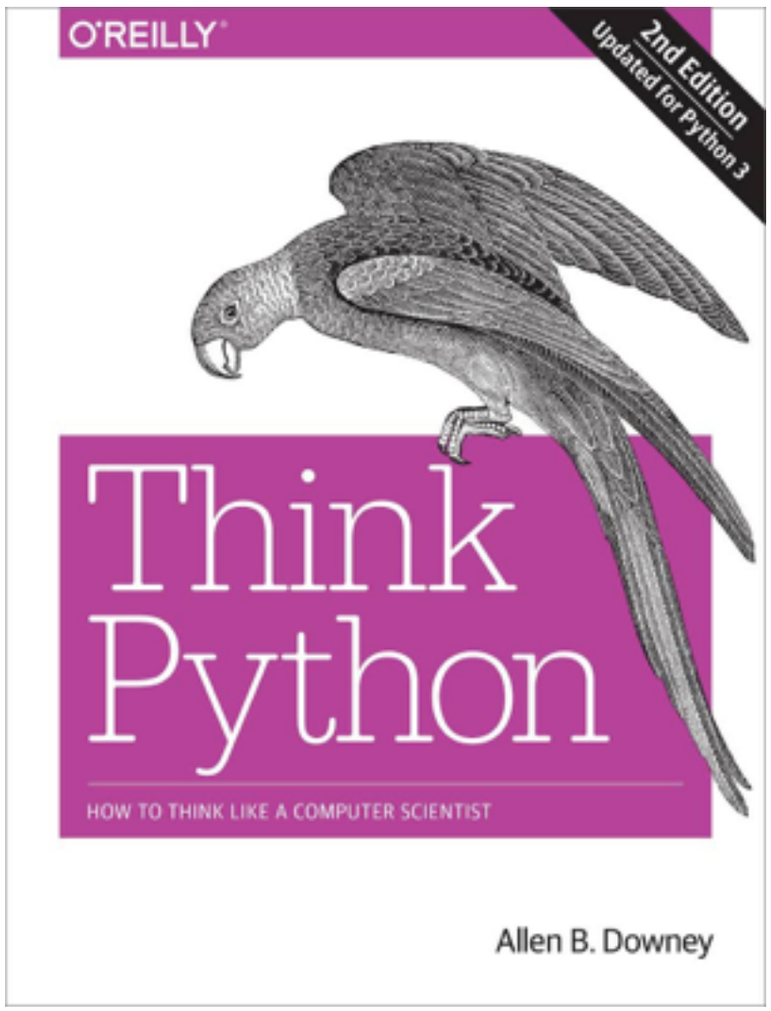
What this course is about?

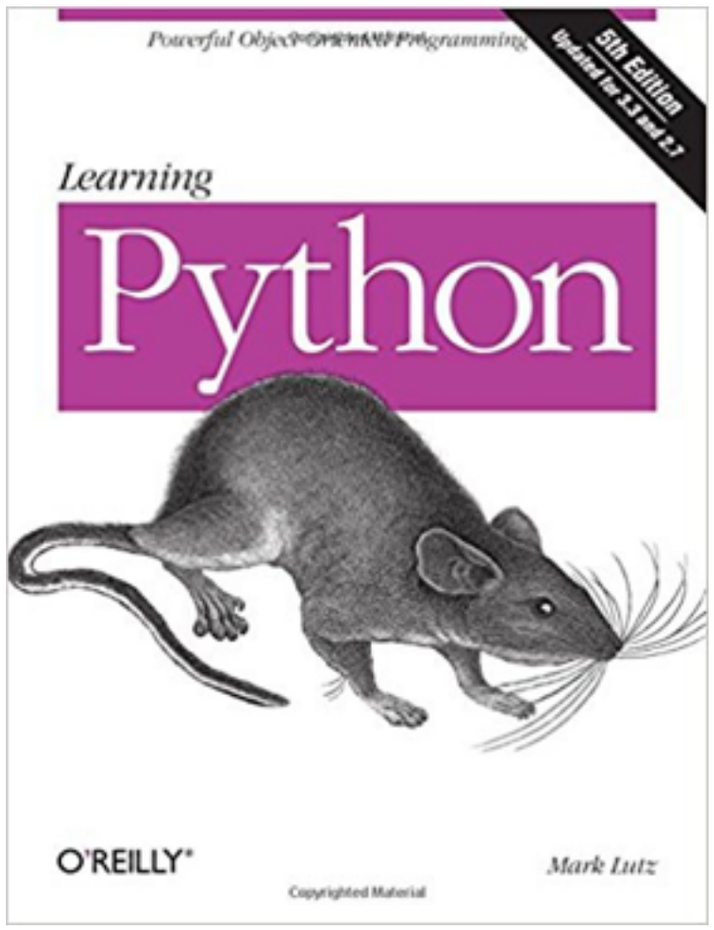
To acquire basic **computational skills** to solve **simple** problems in **science**

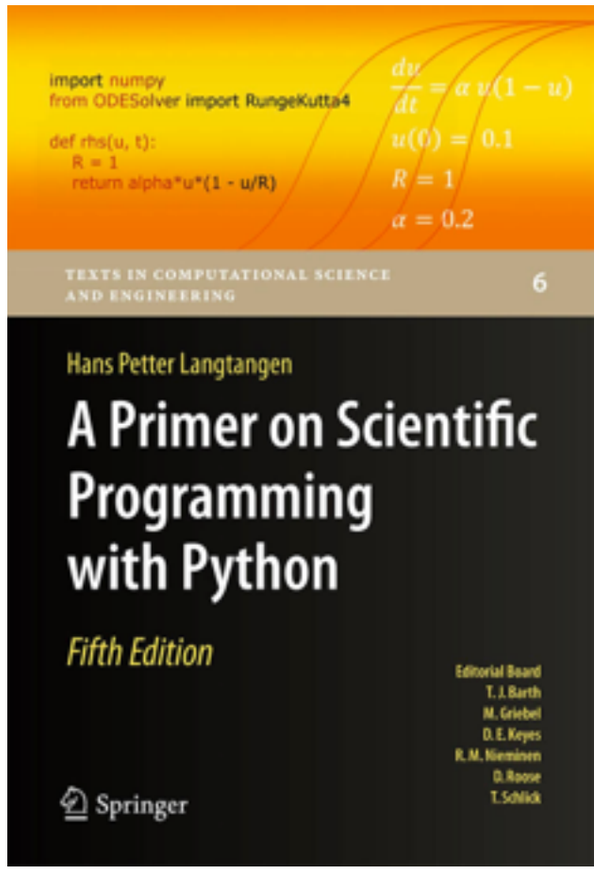
For computation we will use the language known as Python

https://www.python.org/downloads/

Suggested Books for this course

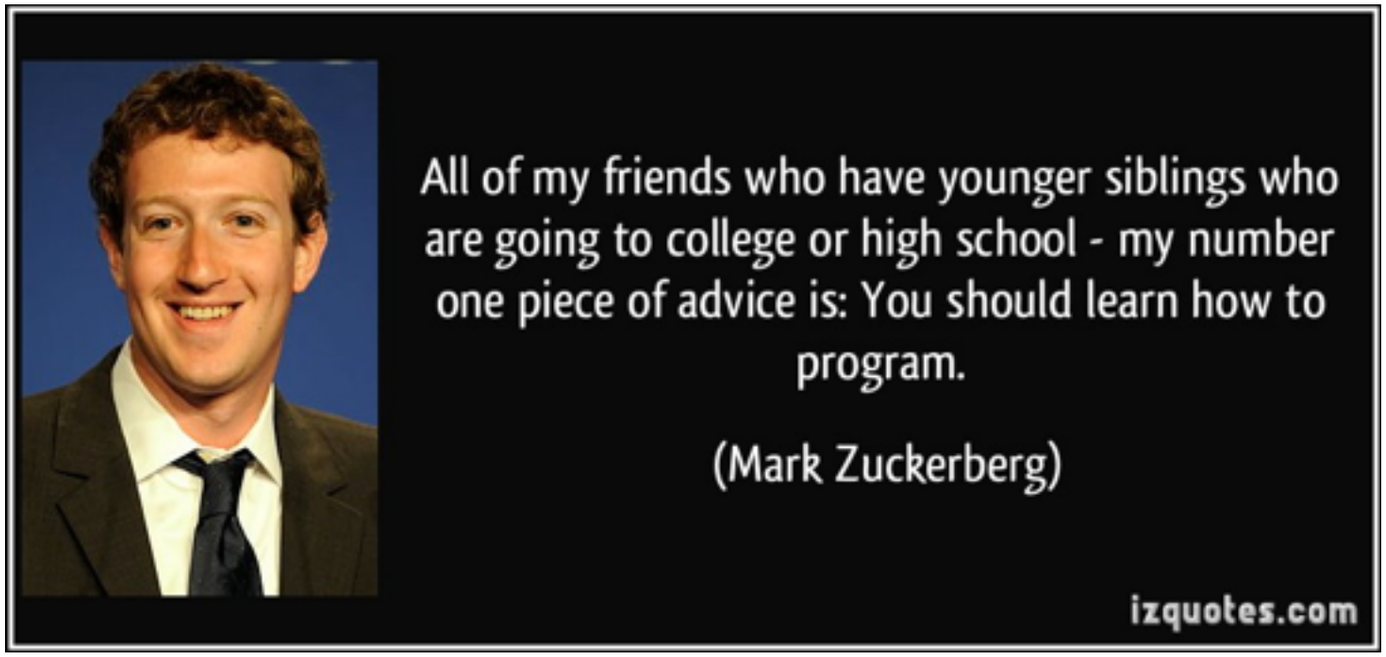


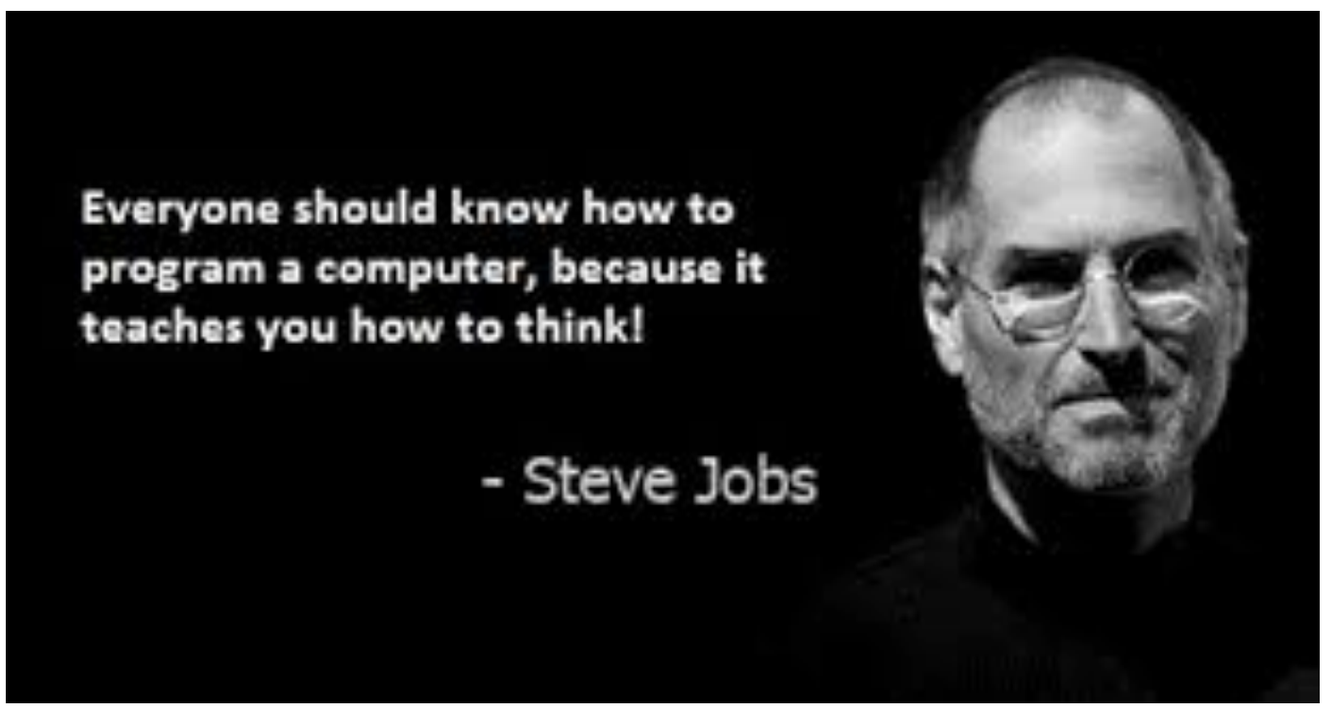




Online sources:
Lectures on Youtube

Some famous quotes





What is Computing?

- Computation is a kind of question answering
- You compute even when you don’t seem to be doing so,
- We compute all the time
- Computer meant humans dates back in 1600

On 18 June 1980, Shakuntala Devi demonstrated the multiplication of two 13-digit numbers—7,686,369,774,870 × 2,465,099,745,779—picked at random by the Computer Department of Imperial College London. She correctly answered 18,947,668,177,995,426,462,773,730 in 28 seconds. This event was recorded in the 1982 Guinness Book of Records.

source: wiki

Johannes Kepler (1571-1630)

Source: Google

Astronomical computations using observational data

Kepler's laws of planetary motion

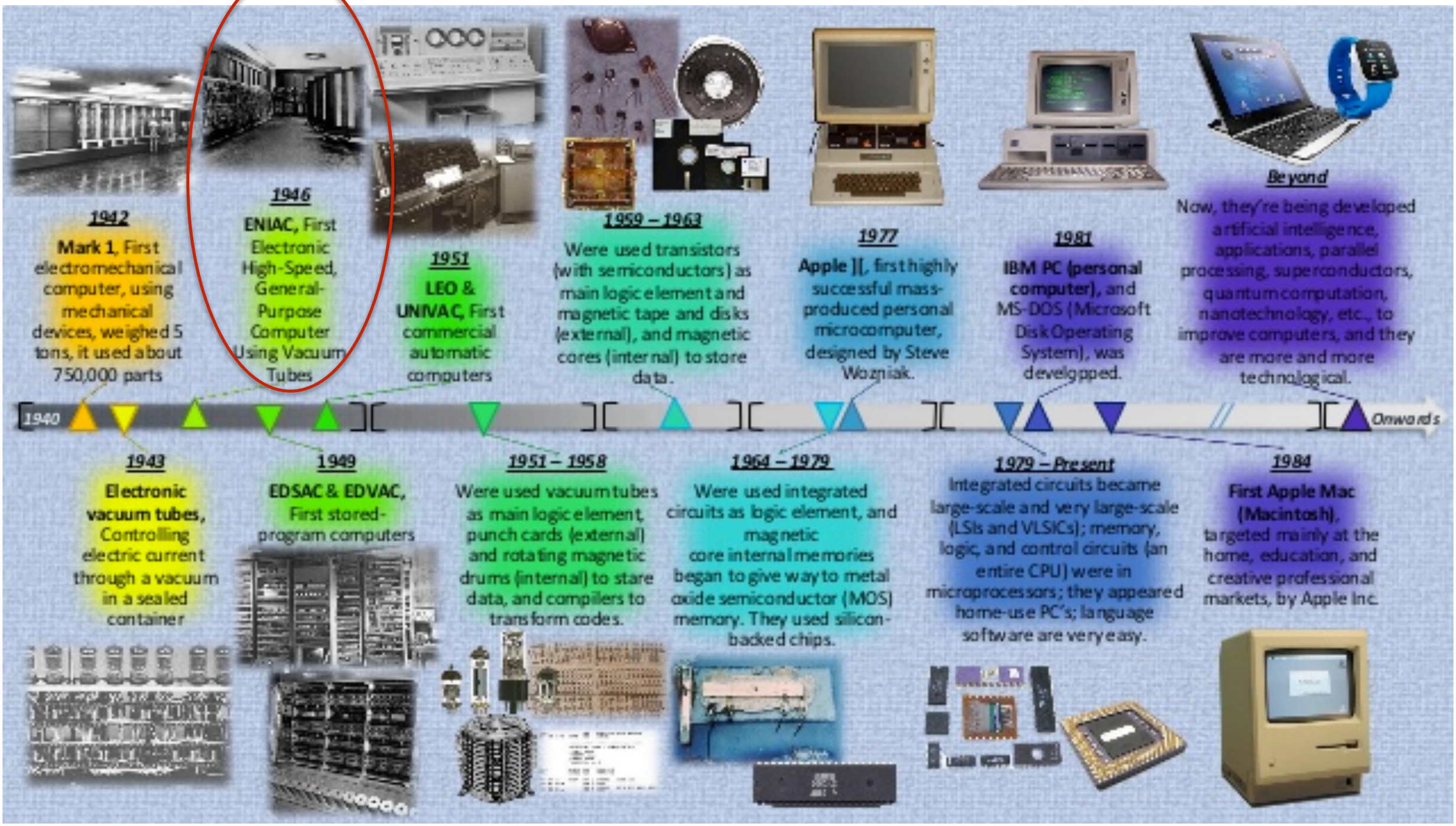
Great Trigonometrical Survey (1802)

Aimed to measure the entire Indian subcontinent with scientific precision

Measurement of the height of the Himalayan giants begun in 1802

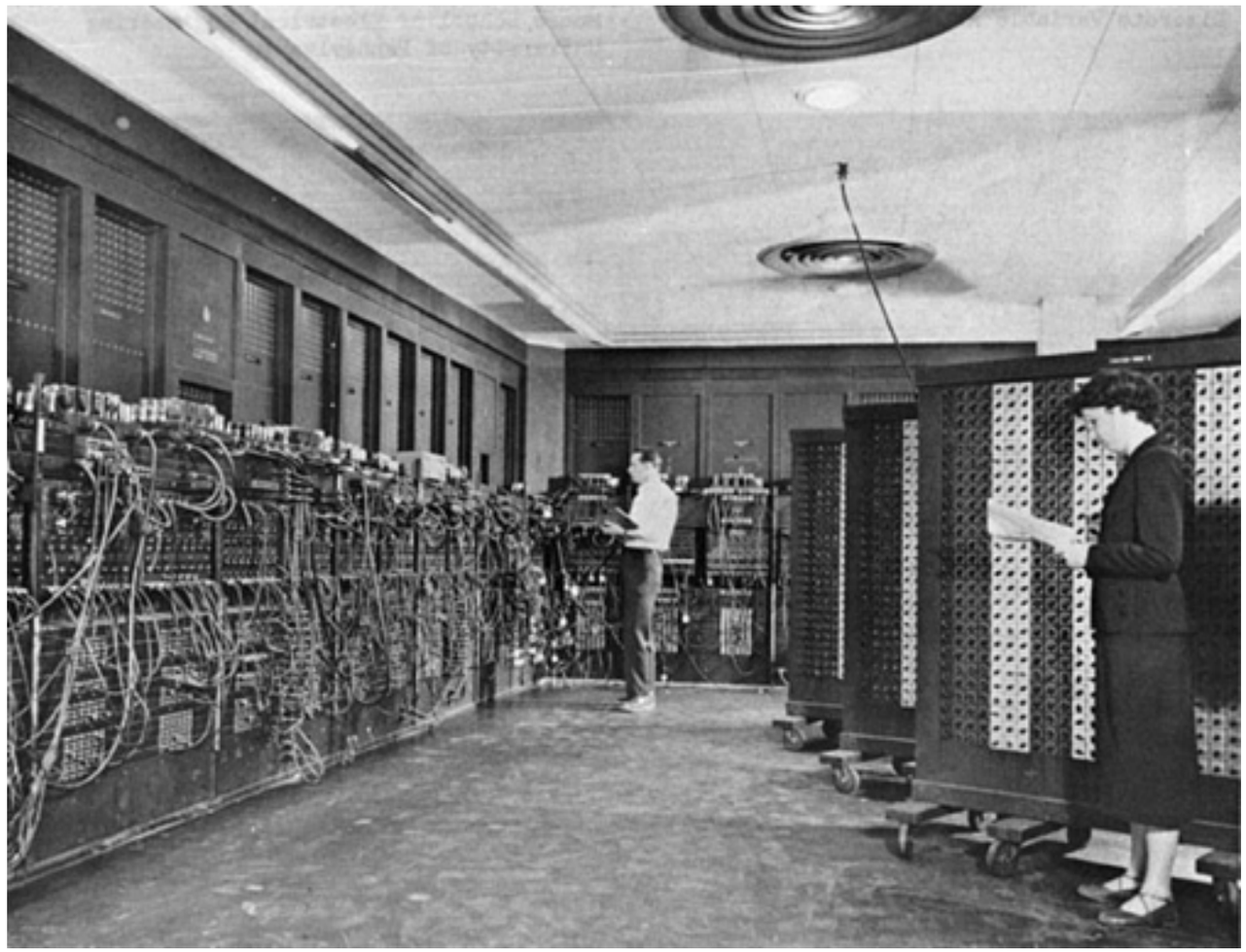
Source: Google

Modern Quantum Age



Source: Google

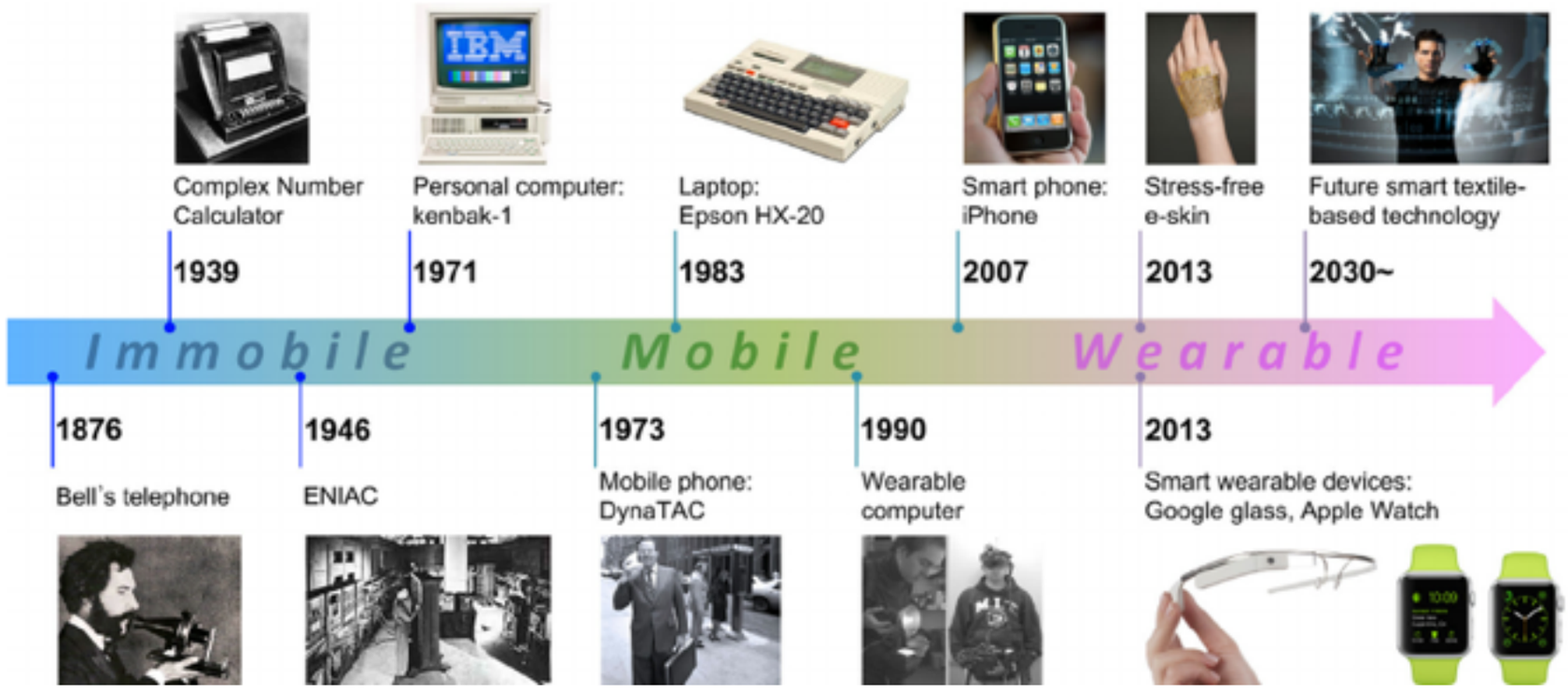
First Fully functional digital computer: ENIAC (Electronic Numerical Integrator and Calculator)



- Invented by J. Presper Eckert and John Mauchly at the University of Pennsylvania
- Began construction in 1943 and completed in 1946.
- It occupied about **1,800 square feet and used about 18,000 vacuum tubes, weighing almost 50 tons.**

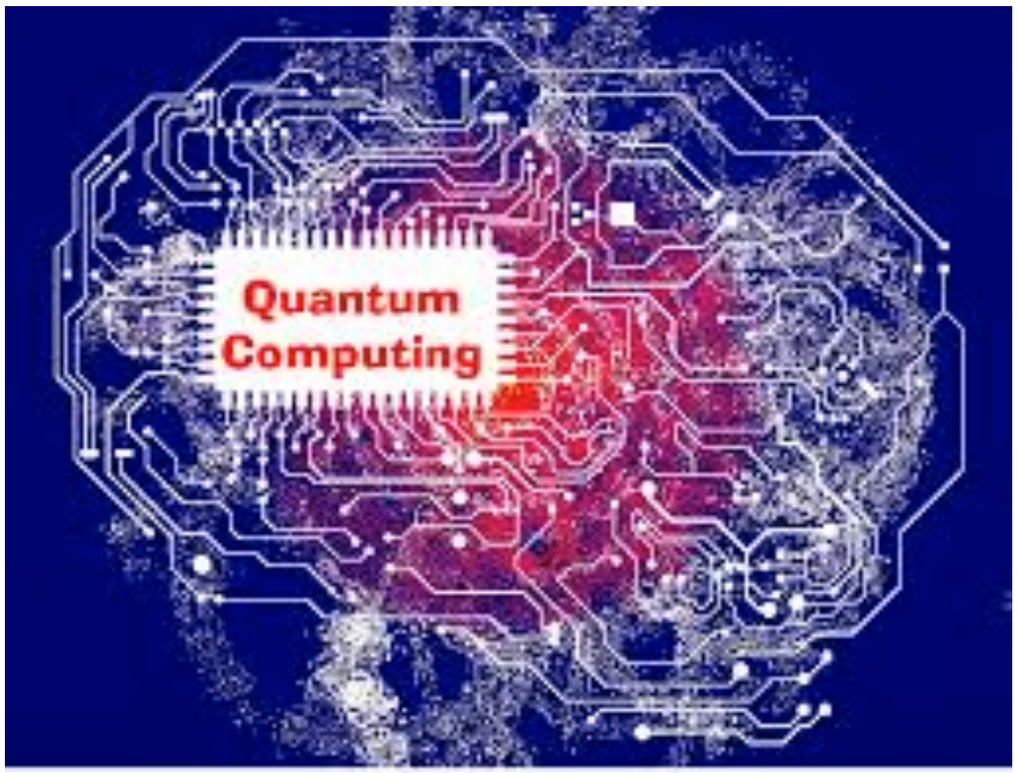
Source: Google

ENIAC to mobile
—> Quick Speed up



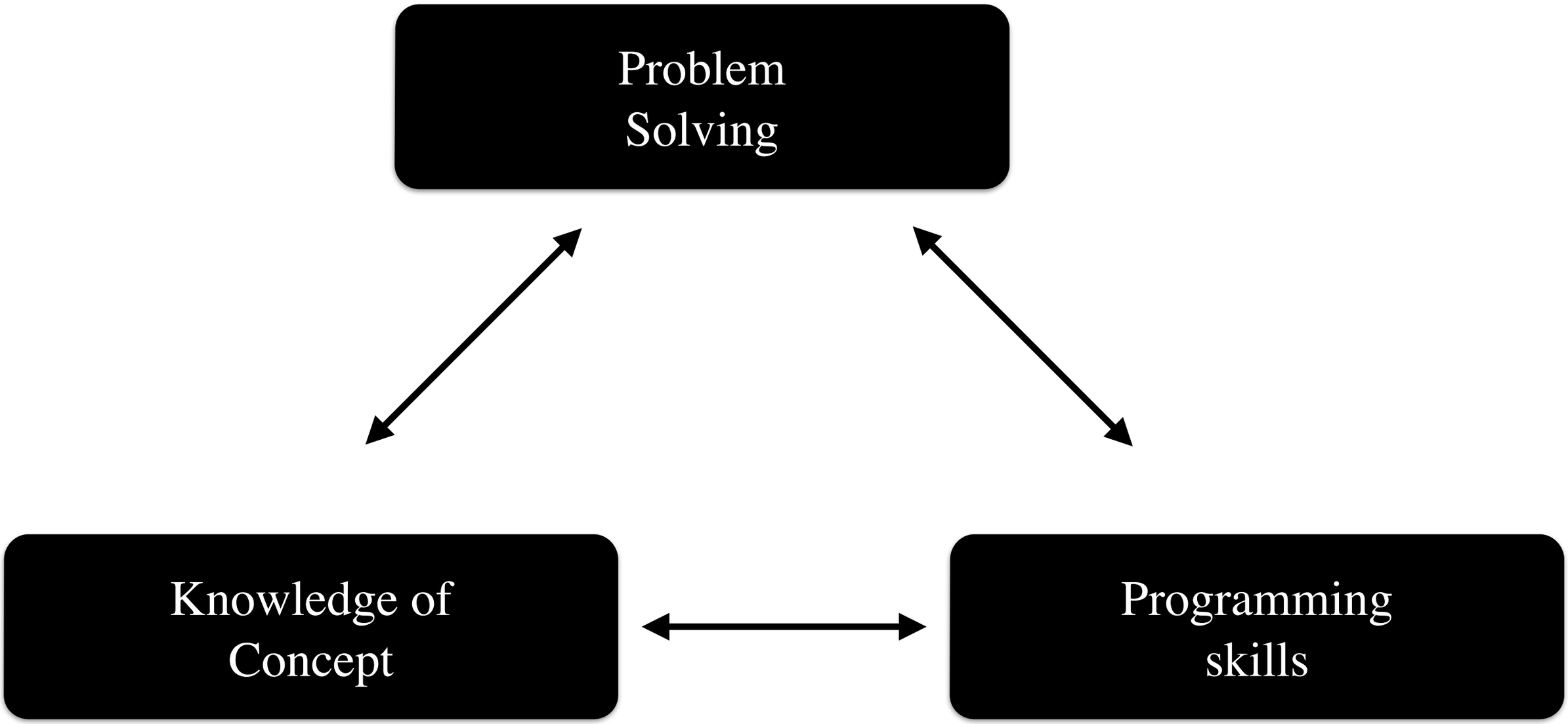
Source: Google

Quantum Computing
—> Exponential speed up



Source: Google

What is Computing?



New to Programming! Practice, Practice….

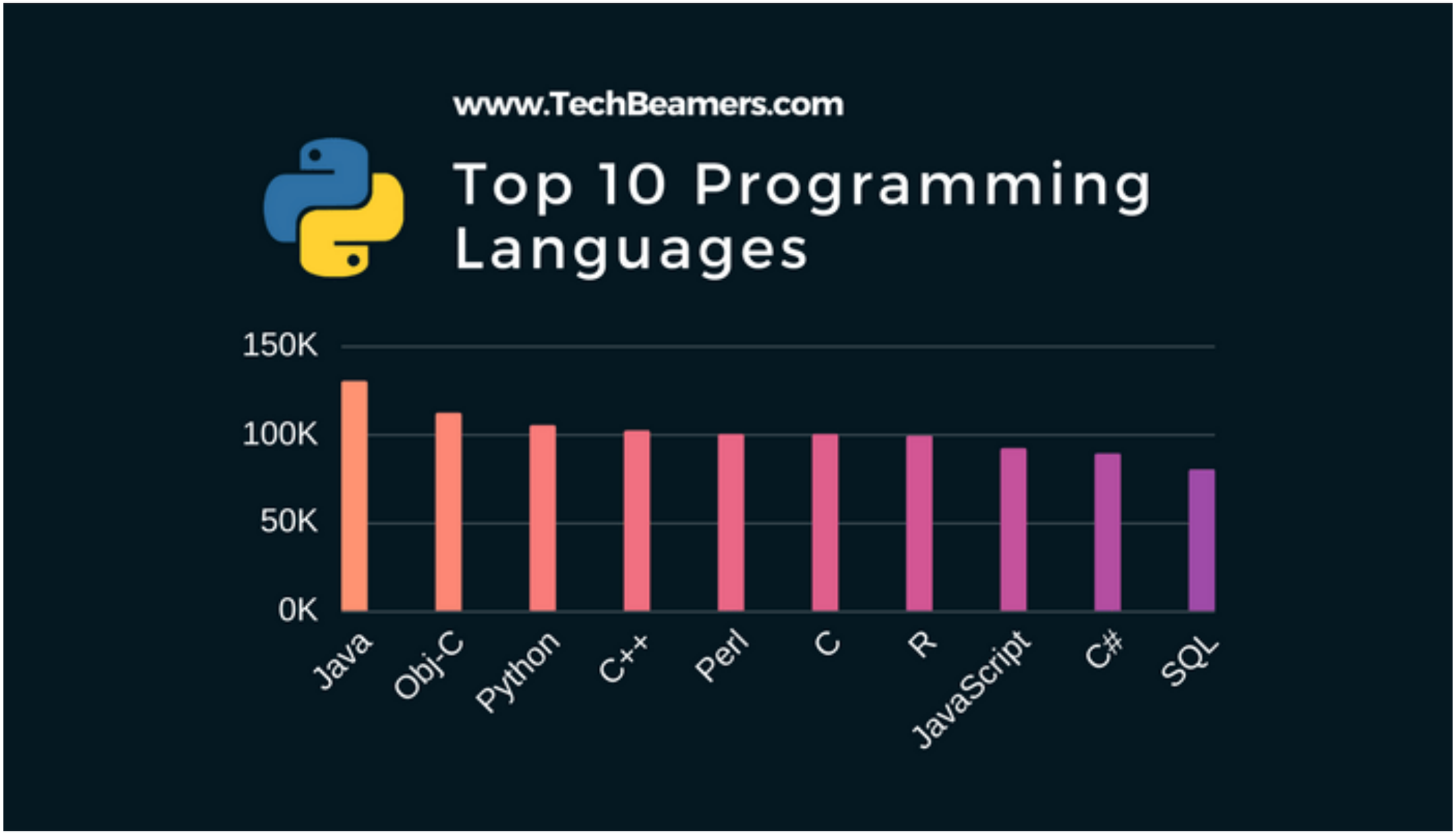
Don’t be afraid: anything happens to computer —-restart :p

What does a computer do?

- Performs Calculations (a billion calculation per second)
- Remembers results (100s of gigabytes of storage)
- What kind of computation?
  built-in to the language
  one that you define as the programmer
- Computers only know what you tell them

Why Python?

- Programming language for beginners
- Simple coding style
- Extremely user friendly
- high-level
- interactive (immediate feedback)
- Scripting language
- Top 5 most popular and fast growing programming language
- 3rd highest earning programming language


www.TechBeamers.com
Top 10 Programming Languages
150K
100K
50K
0K
Java
Obj-C
Python
C++
Perl
C
R
JavaScript
C#
SQL

Python Releases:

- Created in 1989 by Guido Van Rossum (Dutch programmer),
- Python 1.0 released in 1994
- Python 2.0 released in 2000
- Python 3.0 released in 2008
- …..

Van Rossum

(sournce: Wiki)

- Guido Van Rossum wrote Python as a **hobby programming project** back in 1980s. Since then it has grown to become **one of the most polished languages** of the computing world.

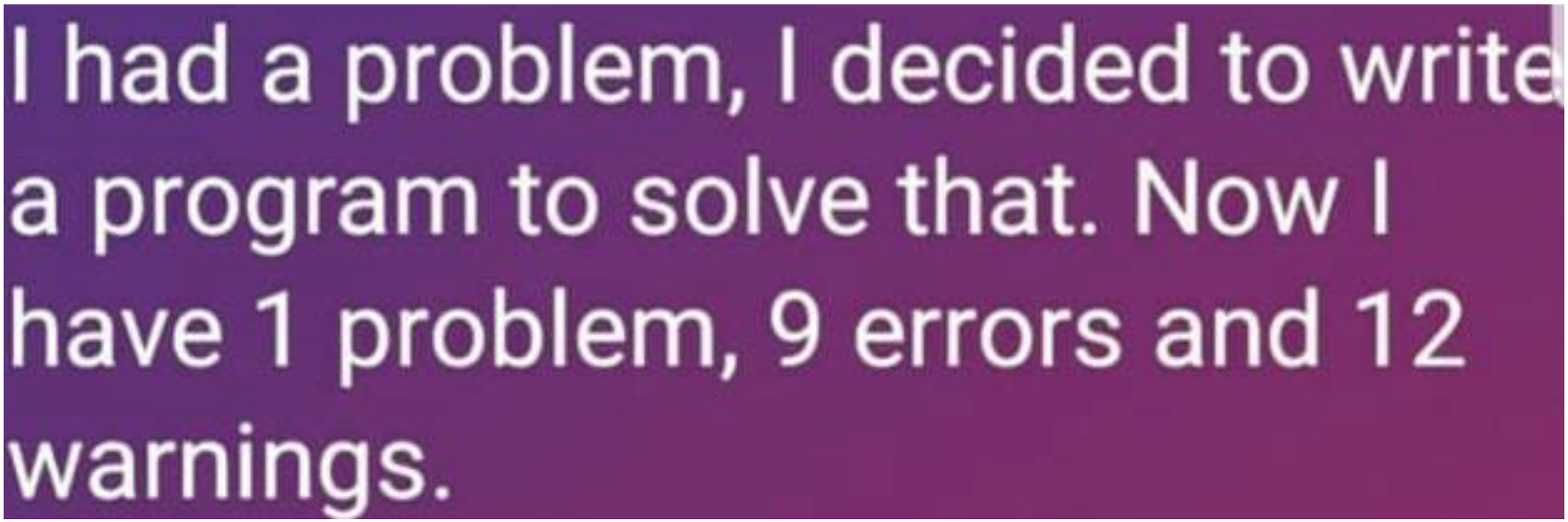
- He was fond of watching the **famous comedy series [The Monty Python's Flying Circus]**. Thus, the name Python struck his mind as not only has it appealed to his taste but also to his target users.

Course Structure

- Fundamentals of Python programming language
- Mathematical Operations
- Python variables and types, control structures, iterations
- Solving simple mathematical/physical problems using python

THANK YOU!

Introduction to Computation
IDC 101



Source: Google

Programming in Python

Program:
A sequence of instructions that specifies how to perform a computation

Examples:
1. Solving a set of linear equations
2. Roots of a polynomial
3. Searching or replacing text in a document

Few basic instructions:

Input `input('Give me a number:')`

Math: perform some basic mathematical operations

Output `print('the value is:', a)`

Repetition

Values and types:

Basic things a program works with : **letter or numbers**

Numbers: integer, floating-point numbers

String: letters

One can always check the type a value is

Syntax: `type(value)`

Examples:

```
>>> type(2)
<class ‘int’>

>>> type(4.0)
<class ‘float’>

>>> type(’12.0')
<class ‘str’>
```

Arithmetic Operators:

+ - * /	addition, subtraction, multiplication, division
%	modulus (remainder)
**	exponentiation

Order of operation

PEMDAS:

P: Parentheses
E: Exponentiation
M: Multiplication
D: Division
A: Addition
S: Subtraction

1+3*4 =?

10+4/2 =?

String Operations

In general, one can't perform mathematical operations on strings

'10'-'2' Not allowed

'char1'/'char2' Not allowed

However + and * operations work for strings

Examples:

```
>>>first= 'IISER'
>>>second='-Pune'
>>> first+second
```

output: 'IISER-Pune'

```
>>> (first+second)*3
```

output: 'IISER-PuneIISER-PuneIISER-Pune'

To know the length of the string:

```
>>>len(variable)
```

Variables and Statements

Variable: A name that refers to a value

Assignment Statement: Stores a value into a variable

Examples: n=11

IISER_Students=1200

A variable that has been given a value can be used in expressions

n+5=16

Print Function:

print: produces text output on the console

Syntax: print ()

Examples:

```
>>> print(5)
5
```

```
>>> print(12.0)
12.0
```

```
>>>print('Hello, World!')
Hello, World!
```

Examples:

```
>>>lecture_no=2
>>>print('This is our', lecture_no, 'nd  lecture in Python')
```

Output:

This is our 2 nd lecture in Python

More about Print Function:

```
print('Single Quotes')

print("double quotes")

print('Can't do this')

print(“can’t do this”)
```

Math Functions

Python has a math **module** that provides most of the familiar math functions

To use these commands, one must write the following at the top of the python program

```
from math import *
```

Command name	Description
`exp(value)`	Exponential
`log(value)`	logarithm base e
`log10(value)`	logarithm base 10
`cos(value)`	cosine in radians
`sin(value)`	sine in radians
`sqrt(value)`	square root
`abs(value)`	absolute value
`max(value1,value2)`	larger of two values
`min(value1,value2)`	minimum of two values

Vocabulary

bug: an error in a program

debugging: The process of finding and correcting bugs

syntax: The rules that govern the structure of a program

execute: to run a statement

Type of errors:

syntax error:

An error in a program that makes it impossible to parse— and therefore impossible to interpret.

```
>>> 17 = n
File "<interactive input>", line 1
SyntaxError: can't assign to literal
```

Type of errors:

semantic error:

An error in a program that makes it do something other than what the programmer intended.

runtime error

An error that does not occur until the program has started to execute but that prevents the program from continuing.

THANK YOU!

New to Programming! Practice, Practice….

Source: Google

Story so-far

Input input('Give me a number:')

Output print('the value is:', a)

Arithmetic Operations with numbers + - * / % **

Operations on Strings

Math Functions from math import *

import math (call using math.)

Formatting in Python

```
>>> x=15.87821

>>> print(format(x,'.5f'))      #rounds the number and shows in decimal
15.87821                        #format with 5 places after decimal.


>>> print(format(x,'>15f'))      #reserves 15 spaces and moves
        15.87821                #them to extreme right


>>> print(format(x,'<15f'))      #reserves 15 spaces and moves
15.87821                        #them to extreme left


>>> print(format(x,'^15f'))      #reserves 15 spaces and centers it
   15.87821                     #in those spaces
```

```
>>> print(format(x,'10.5f'))
15.87821
```

```
#Reserves 10 spaces. Rounds the number
# and shows in decimal format with
# 5 places after decimal.
```

You can display the number in one of the foll formats :

```
f -> decimal format
e -> scientific notation
d -> integer format
s -> string
```

Selection (if/else) and Repetition

Decision making

Want to execute a code only if a certain condition is satisfied.

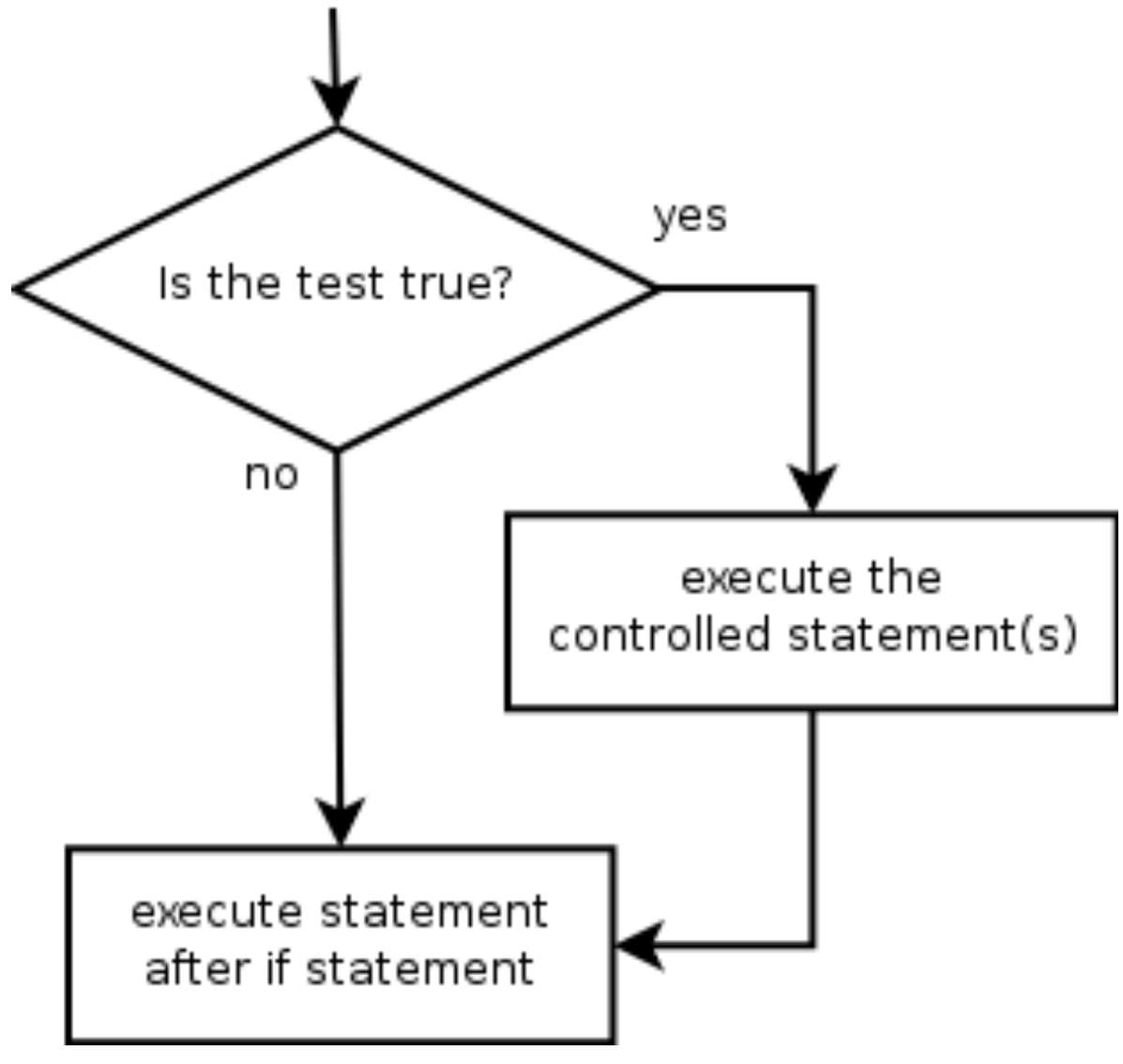
if

if statement:

Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped

Syntax:

if condition :
statements



Program to divide two numbers

```
num = float ( input('Give numerator : ') )
den = float ( input('Give denominator : ') )

print (' ')
print('Numerator : ', num)
print('Denominator : ', den)
print (' ')


if den != 0.0 :
    y=num/den
    print(y)
```

Incorrect indentation will result into IndentationError.

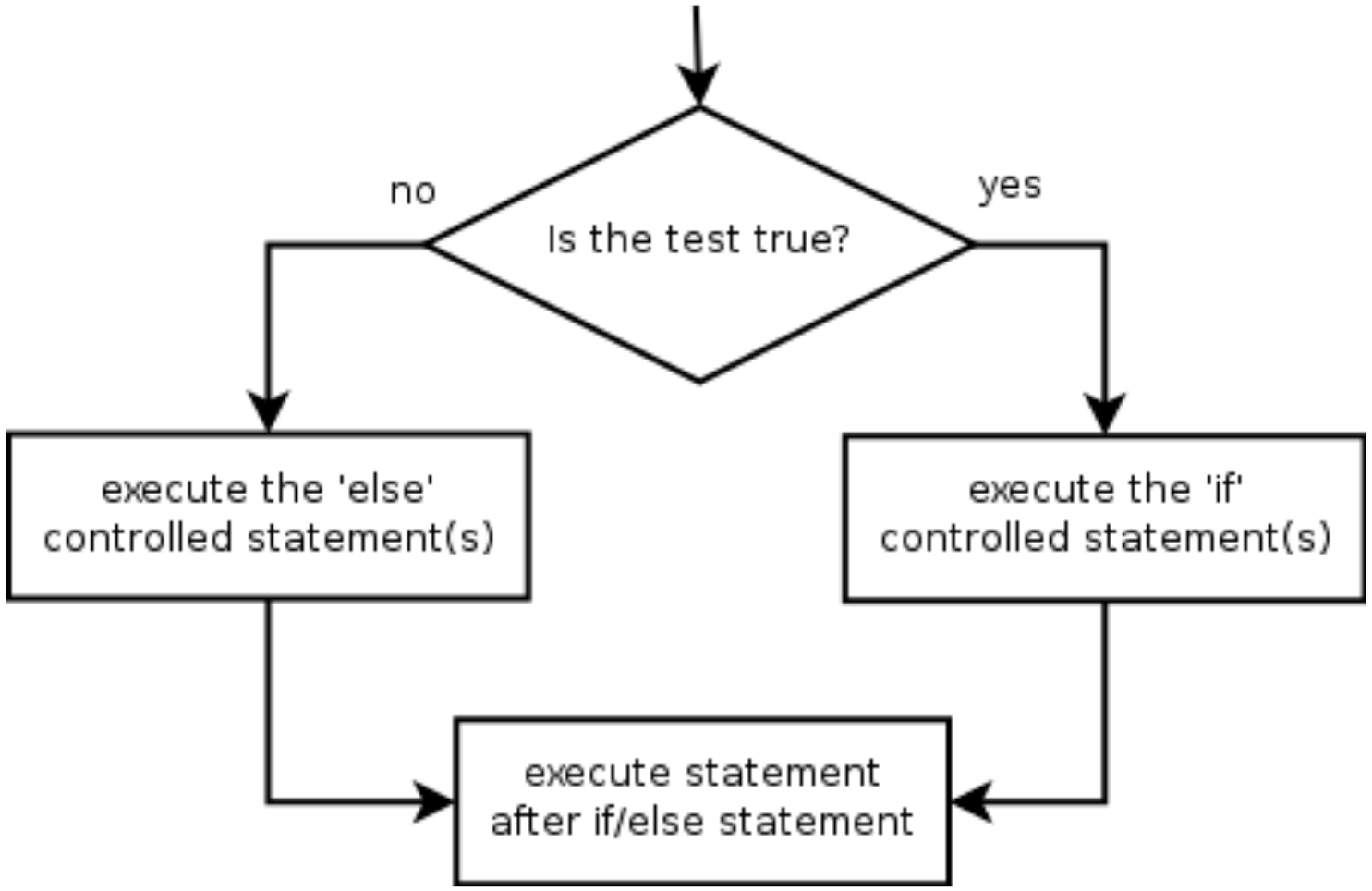
if/else

if/else statement:

Executes one block of statements if a certain condition is True, and a second block of statements if it is False.

Syntax:

```
if condition:
    statements
else:
    statements
```



Ex: Program to divide two real numbers (version 2)

```
num = float ( input('Give numerator : ') )
den = float ( input('Give denominator : ') )

print (' ')
print('Numerator : ', num)
print('Denominator : ', den)
print (' ')
```

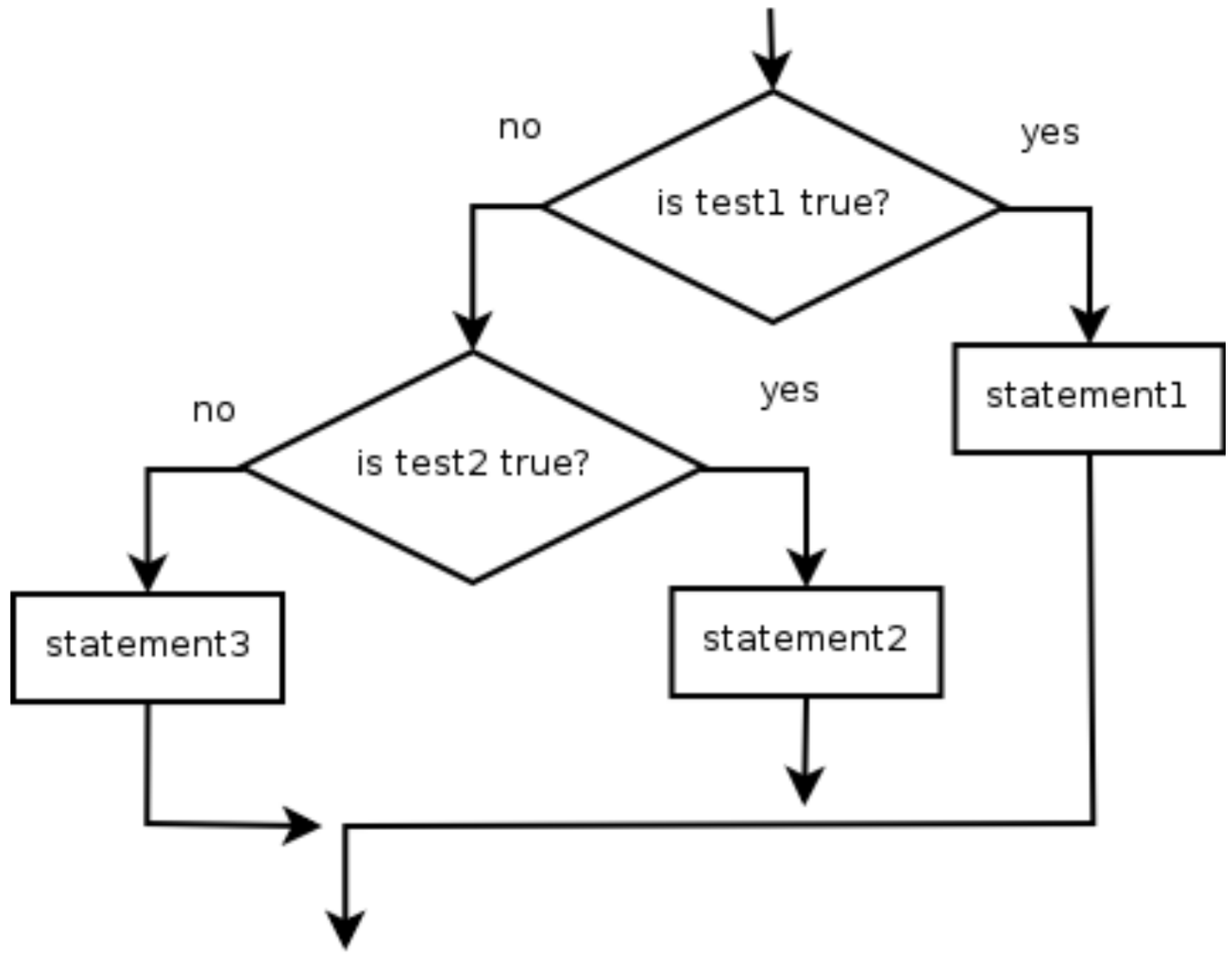
```
if den == 0.0 :
    print ('WARNING : Division by zero')
else :
    y=num/den
    print(y)
```

elif

Multiple conditions can be chained with `elif` ("else if"):

Syntax:

```
if condition:
    statements
elif condition:
    statements
else:
    statements
```



Ex: Takes time as input and writes if it is morning, afternoon or evening hours.

```
time = float( input('Give present time: ') )

if (time>0) and (time <=12):
    print('Morning hours’)
elif (time>12) and (time<=18):
    print('Afternoon hours’)

else:
    print('Evening hours')
```

Ex: Takes time as input and writes if it is morning, afternoon or evening hours.

```
time = float( input('Give present time: ') )

  if ( time > 0 ) and ( time <= 12 ) :
     print('Morning hours')
  else:
  if ( time > 12 ) and ( time <= 18 ) :
     print ('Afternoon hours')
  else:
     if ( time > 18 ) :
        print('Evening hours')
```

Logic

Many logical expressions use ***relational operators***:

Operator	Meaning	Example	Result
==	equals	`1 + 1 == 2`	`True`
!=	does not equal	`3.2 != 2.5`	`True`
<	less than	`10 < 5`	`False`
>	greater than	`10 > 5`	`True`
<=	less than or equal to	`126 <= 100`	`False`
>=	greater than or equal to	`5.0 >= 5.0`	`True`

Logical expressions can be combined with ***logical operators***:

Operator	Example	Result
and	`9 != 6 and 2 < 3`	`True`
or	`2 == 3 or -1 < 5`	`True`
not	`not 7 > 0`	`False`

```
>>> x = 10
>>> y = 12
```

```
>>> print('x > y  is',x>y)
```
Output: x > y is False

```
>>> print('x < y  is',x<y)
```
Output: x < y is True

```
>>> print('x == y is',x==y)
```
Output: x == y is False

Source: Google

Repetition

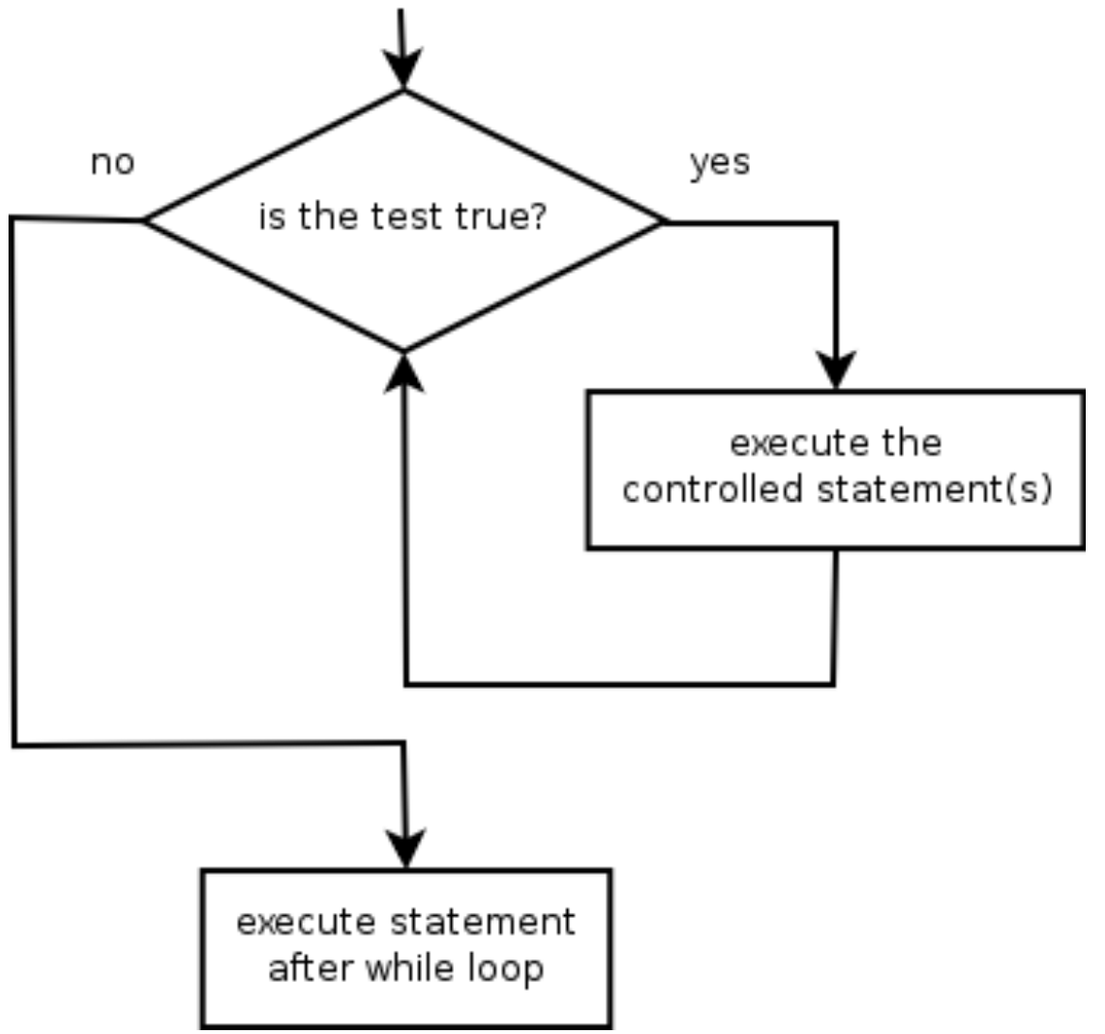
while

while loop: Executes a group of statements as long as a condition is True.

- good for *indefinite loops* (repeat an unknown number of times)

Syntax:

```
while condition:
    statements
```



Example:

```
number = 1
 while number < 100:
      print(number)
      number = number * 3
```

Output:

```
1 3 9 27 81
```

range

The range function specifies a range of integers

range (start, stop) *The integers between start (inclusive) and stop(exclusive)*

It can also accept a third value specifying the change between the values

range (start, stop, step) *The integers between start (inclusive) and stop(exclusive) by steps*

Example:

```
for x in range(1,6,2):
    print(x)
```

Output:

```
1 3 5
```

```
x=range(6)
```

Output:

```
0 1 2 3 4 5
```

List

list: a sequence of values, a collection which is ordered and changeable

The values can be of any type.

The values in a list are called elements or items

Python lists are written with square brackets.

```
my_list = []                    # it creates an empty list
```

```
my_list = [1, 2, 3]             # creates list of integers
```

```
my_list = [1, "Hello", 3.4]     # list with mixed datatypes
```

Nested List: A list within another list

```
zs = ["hello", 2.0, 5, [10, 20]]
```

Access items from list

```
>>> my_list=[1,"Hello",3.4]

>>> my_list[0]
 1

>>> my_list[1]
'Hello'

>>> my_list[2]
 3.4
>>> my_list[3]
IndexError: list index out of range

>>> len(my_list)
 3
```

We can add one item to a list using append() method or add several items using extend() method.

```
>>> my_list.append('World')

>>> my_list
[1, 'Hello', 3.4, 'World']
```

List

```
>>> my_list.extend(['earth','water'])

>>> my_list
[1, 'Hello', 3.4, 'World', 'earth', 'water']


>>> my_list.insert(1,'Bye')                    # Insert ‘Bye’ at pos 1, shift other items up
>>> my_list
[1, 'Bye', 'Hello', 3.4, 'World', 'earth', 'water']


>>> my_list.remove('Bye')                      # Remove ‘Bye’ at pos 1
>>> my_list
[1, 'Hello', 3.4, 'World', 'earth', 'water']
```

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

List Slices

```
>>> a_list = ["a", "b", "c", "d", "e", "f"]

>>> a_list[1:3]
['b', 'c']
>>> a_list[:4]
['a', 'b', 'c', 'd']
>>> a_list[3:]
['d', 'e', 'f']
>>> a_list[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

List Operations

+ operation

```
>>> a=[1,2,3]
>>> b=[4,5,6]

>>> a+b

[1, 2, 3, 4, 5, 6]
```

*** operation**

```
 >>> [1]*4

 [1, 1, 1, 1]

>>> [1,2,3]*3

[1, 2, 3, 1, 2, 3, 1, 2, 3]

>>> list(range(0,10,2))

 [0, 2, 4, 6, 8]
```

The for loop

for **loop**: A for loop is used for iterating over a sequence (for example: a list).

With the for loop we can execute a set of statements,
once for each item in a list.

Syntax:

```
for variable name in values:
    statements
```

Example:

```
for x in range(1,6):
    print(x)
```

Output:

```
1 2 3 4 5
```

```
for x in range(1,-6, -2):
    print(x)
```

Output:

```
1 -1 -3 -5
```

The for loop: example

```
fruits = ["apple", "banana", “cherry"]
for x in fruits:
     print(x)
```

Output:

```
apple
banana
cherry
```

```
fruits = ["apple", "banana", “cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

Output:

```
apple
banana
```

The for loop with break statement

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

Output:

```
apple
```

Even strings are iterable objects, they contain a sequence of characters:

```
for x in "banana":
    print(x)
```

Output:

```
b
a
n
a
n
a
```

More examples

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

output:

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

THANK YOU!

Quiz1 on19th September

Batch-3: 11:30 to 12:30 pm
Batch-1: 1:45 to 2:45 pm
Batch-2: 3:15 to 4:15 pm
Batch-4: 4:45 to 5:45 pm

Important instructions related to the quiz.

1. Note that this test will not be repeated or rescheduled under any circumstances if you are absent for the test.

2. This test will cover everything upto assignment number 4 (31st August).

3. During this test, you will not have access to internet, notes, books, previously written programs or any other help.

4. Test will be in the form of 2-3 questions. You are expected to write a program and show the program and the output by executing it. Time allowed is 60 minutes

5. If you need additional practice sessions in the computer, feel free to use the computer lab. Also, you can consult any of the instructors or the TAs for doubts.

Type of errors:

Syntax error:

An error in a program that makes it impossible to parse— and therefore impossible to interpret.

```
>>> 17 = n
File "<interactive input>", line 1
SyntaxError: can't assign to literal
```

Semantic error:

An error in a program that makes it do something other than what the programmer intended.

Runtime error

An error that does not occur until the program has started to execute but that prevents the program from continuing.

These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened.

Error

Which of the following is a semantic error?

(A) Attempting to divide by 0.

(B) Forgetting a colon at the end of a statement where one is required.

(C) Forgetting to divide by 100 when printing a percentage amount.

What is the output?

```
n=int(input('Give me a number:'))
i=1
while i<=n:
        i=i+1
        i=i-1
        print(i)
print(i)
```

Break statement

The break statement causes a program to break out of a loop.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

Output:

```
apple
```

```
number = 0

for number in range(10):
   number = number + 1

   if number == 5:
      break    # break here

   print('Number is ` number)

print('Out of loop')
```

Output:

```
Number is  1
Number is  2
Number is  3
Number is  4
Out of loop
```

More examples: break

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
     if y == 'banana':
        break
     print(x, y)
```

Output:

```
red apple
big apple
tasty apple
```

Continue statement

Gives you the option to skip over the part of a loop where an external condition is triggered but to go on to complete the rest of the loop.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
print(x)
```

Output:

```
apple
cherry
```

More examples

```
number = 0

for number in range(10):
    number = number + 1

    if number == 5:
        continue    # break here

    print('Number is', number)

print('Out of loop')
```

Output

```
Number is 1
Number is 2
Number is 3
Number is 4
Number is 6
Number is 7
Number is 8
Number is 9
Number is 10
Out of loop
```

Functions

Function is a group of related statements that perform a specific task

You can pass data, known as parameters, into a function.

A function can return data as a result.

Python gives you many built-in functions like print(), input(), abs() etc.

But you can also create your own functions.

These functions are called **user-defined functions.**

function syntax

Syntax:

```
def my_function(parameters):
    statements
    return [expression]
```

Note the syntax to define a function:

1. The def keyword
2. Followed by the function's name,
3. The arguments of the function are given between brackets followed by a colon.
4. The function body ;
5. and return object for optionally returning values.

Examples

Write a program to compute the average of three numbers

```
n1=float(input('Give me the first number:'))
n2=float(input('Give me the second number:'))
n3=float(input('Give me the third  number:'))


#---------------Here is the the function ---------------
def average(x1,x2,x3):
    z=(x1+x2+x3)/3
    return z


#---------Calling the  function -------

value=average(n1,n2,n3)
print('The average is:', value)
```

What is wrong with the following function definition:

```
def addEm(x, y, z):
    return x + y + z
    print('the answer is', x + y + z)
```

(A) You should never use a print statement in a function definition.
(B) You should not have any statements in a function after the return statement. Once the function gets to the return statement it will immediately stop executing the function.
(C) You must calculate the value of x+y+z before you return it.
(D) A function cannot return a number

What will the following function return?

```
def addEm(x, y, z):
    print(x + y + z)
```

Introduction to planning

An algorithm is like a recipe in a cook book. It is a step by step set of instructions that the computer will have to follow to solve a problem or complete a task.

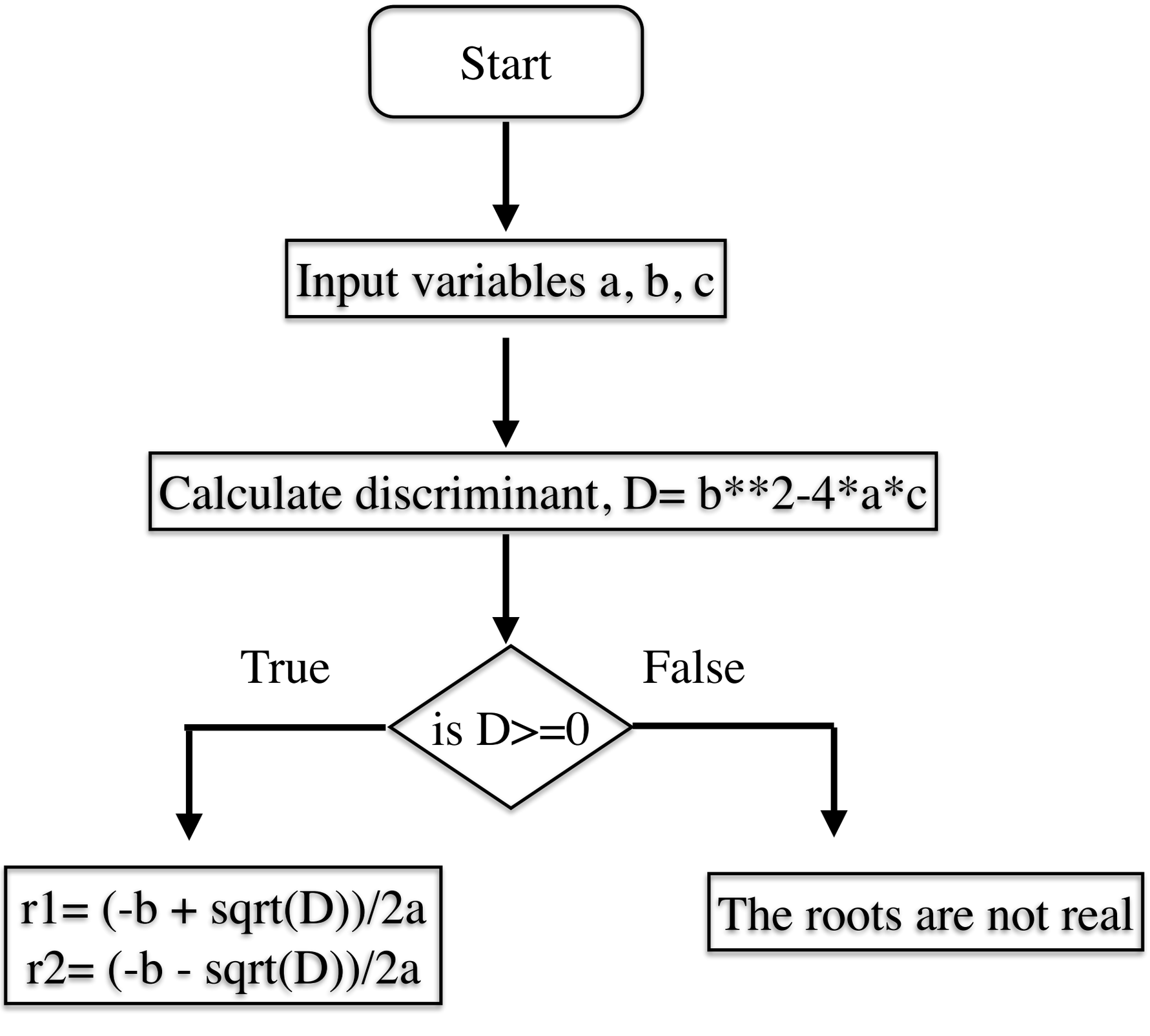
To design an algorithm/ program you can draw a flowchart or write pseudo-code.

It is important to be able to plan code with the use of flowcharts.

Even though you can code without a plan, it is not good practice to start coding without a guide to follow.

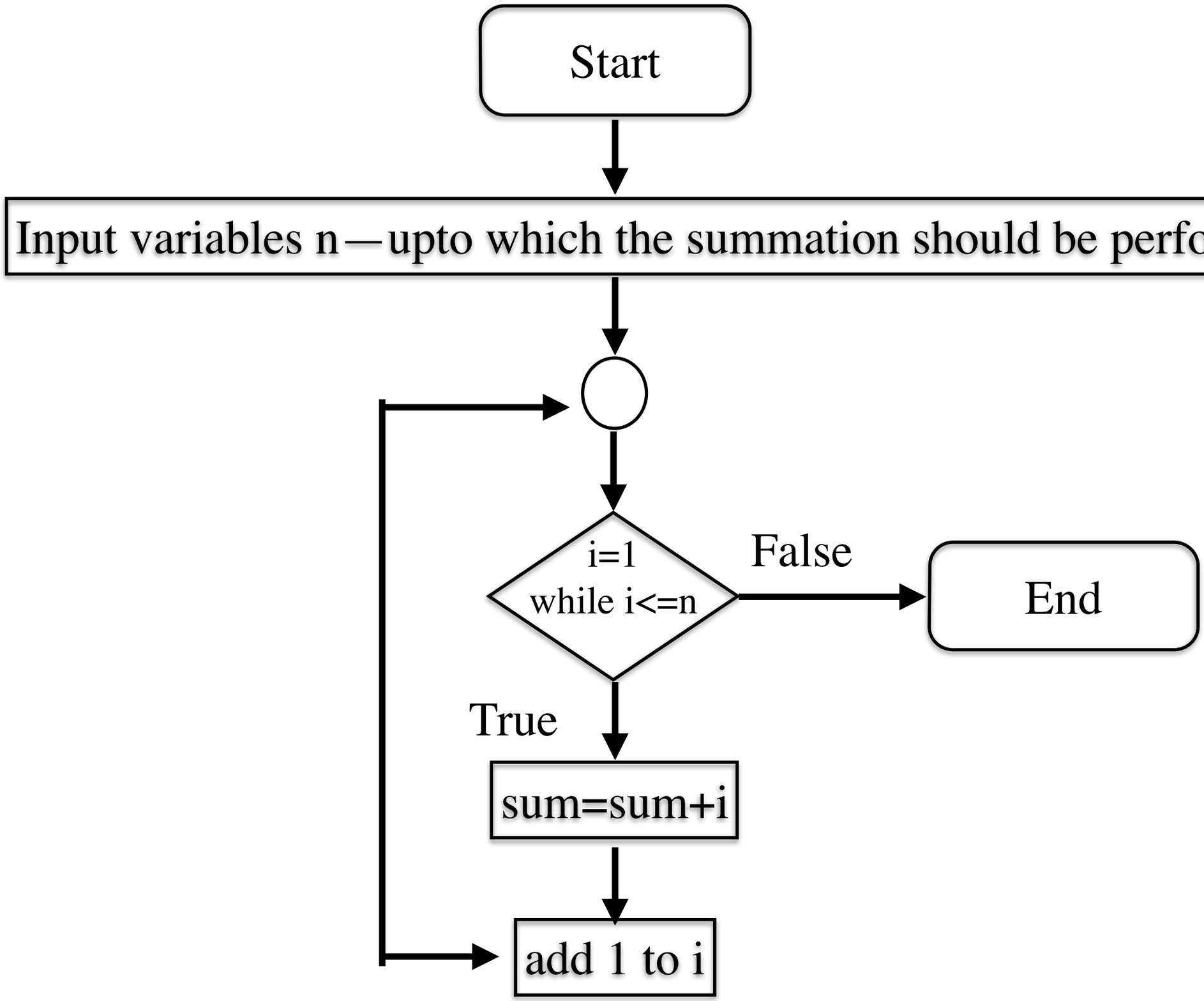
Flowchart

Draw a flowchart to find the roots of a quadratic equation



Flowchart: Continue

Draw a flowchart to find the sum from 1 upto given integer number by the user



#This program computes the exponential series using a factorial function

```
x=float(input('Give me the value to compute the exp function:'))
sum1=1.0
nmax=10000

#---------------Here is the the function ---------------
def fac(a):
    temp=1
    for i in range(1,a+1):
       temp=temp*i
    b=temp
    return b

#--------HERE is the main part of the program ----------
for i in range(1,nmax):
      y=fac(i)        # here i am calling the function
      nsum1=sum1
      sum1=sum1+((x**i)/y)
      if abs(nsum1-sum1)<1e-5:
         print('convergence achieved',i)
         break

print('exp(',x,')=', sum1)
```

THANK YOU!