Flow of execution

Flow of execution

The *flow of execution* is the order in which program statements are executed.

Unless specified otherwise, program execution always begins from the first line. Statements are executed one at a time, top down.

Flow of execution

The *flow of execution* is the order in which program statements are executed.

Python provides three constructs to control the flow of execution.

- Functions
- Conditionals
- Iteration

Iteration

Loop constructs allow a program to repeat executing sequences of instructions under certain conditions.

The most basic form of loop construct is the *while* statement:

```
while <condition>:
  <statements>
```


Iteration

The most basic form of loop constructs is the *while* statement:

```
while <condition>:
  <statements>
```

- `<condition>` is a *boolean* expression.
- `<statements>` can be any sequence of statements.

`if` vs. `while`

In the *if* statement

```
if <condition>:
  <statements>
```

`<statements>` is executed *just once* in case `<condition>` is `True`.

if vs. while

In the *while* statement

```
while <condition>:
  <statements>
```

<statements> is repeatedly executed *over and over again* as long as <condition> is True. The loop terminates only when <condition> becomes False.

if vs. while

In the *while* statement

```
while <condition>:
  <statements>
```

just like the *if* statement, if <condition> is False to start with, then <statements> is never executed.

Meaningful loops

```
while <condition>:
  <statements>
```

In order for this loop to have any value, the value of <condition> must become from True to False at some point while <statements> is repeatedly executed.

Iteration

Example.

```
i = 10
while i > 0:
  print i,
  i = i - 1
```

This simply means that, as long as `i > 0`, print the value of `i` and decrement `i` by `1`.

Iteration

Example.

```
i = 10
while i > 0:
  print i,
  i = i - 1
```

Output: `10 9 8 7 6 5 4 3 2 1`

Iteration

Example.

```
i = 10
while i > 0:
  print i,
```

Output: `10 10 10 10 10 10 10 10 10...`

Iteration

Example.

```
i = 10
while i > 0:
  print i,
  i = i - 1
```

In order for this to terminate, we need a statement that changes the value of `i > 0` from `True` to `False`.

Iteration

Example.

```
sum = 0
n = input('Enter a number: ')
while n > 0:
  sum = sum + n
  n = input('Enter a number: ')
print 'The sum of is', sum
```

In this example, there is no statement that explicitly changes the value of `n > 0` from `True` to `False`.

Iteration

Example.

```
sum = 0
n = input('Enter a number: ')
while n > 0:
  sum = sum + n
  n = input('Enter a number: ')
print 'The sum of is', sum
```

The user gets to decide when to terminate by entering a value that makes `n > 0` from `True` to `False`.

Computing is old

Computing is critical to these technologies, but computing is actually a really old idea…

At the heart of computing is this notion of

algorithms

Algorithm

A precise sequence of simple instructions that can be automated.

Examples.

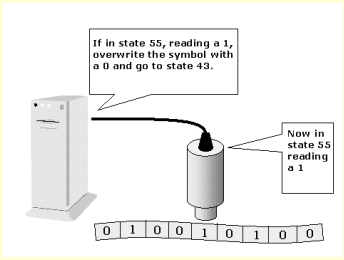
- The procedure to add any given two large numbers.
- The procedure (recipe) to cook clam chowder.
- …

Turing machine (1937)

The Turing machine is the earliest formal theoretical model of a computer.

- A very simple device that manipulates symbols (0's and 1's) on a tape.
- Despite its simplicity, its computational power is equivalent to any computer.

Turing machine (1937)



http://www.lambdaassociates.org/

Hardware layers of a computer

A computer may look very complex, but its hardware is comprised of layers of simple components.

- bits
- switches
- gates
- circuits
- the processor and memory

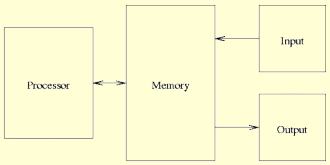
The processor and memory

The processor and memory are the most critical components of a computer. Both are made out of circuits.

- A *processor* executes computer programs (which are basically algorithms written in a computer language).
- *Memory* stores a large amount of information (in bits). Programs as well as data are stored in memory.

von Neumann model

All computers share the following basic design called the *von Neumann* model (named after the Hungarian/American mathematician John von Neumann (1903–1957)).



Instruction cycle

Under this model, the processor executes a program by repeating the following cycle.

1. fetch — reads an instruction from the memory
2. decode — decodes the instruction to figure out its meaning.
3. execute — performs the task specified in the instruction.
4. store — stores the result of the above execution in the memory.

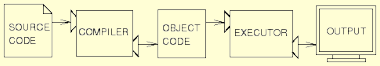
High-level languages

In practice, programs are written in *high-level languages* that can be understood by both human and computers.

Such programs are then translated to binary codes (in a machine language).

A compiler is a software system to perform such translation.

Compiler



- Translates a program into binary/object code completely before execution.
- Program execution is done by running binary/object code.

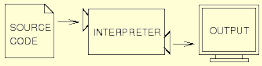
High-level languages

In practice, programs are written in *high-level languages* that can be understood by both human and computers.

Such programs are then translated to binary codes (in a machine language).

An *interpreter* provides another way to perform such translation.

Interpreter



- Translates and executes a program one line at a time.
- Unlike compilers, programs must be translated every time they are executed.
- Inefficient in general, but convenient!