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

Binary notation

For example, numbers are represented in 0's and 1's in the following way.

0	¥	0
1	¥	1
2	¥	10
3	¥	11
4	¥	100
5	¥	101
	...	

Basics Elements of a Computer

Bits

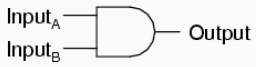
A *bit* is the smallest unit of information used in computers. A bit is either 0 or 1.

Information of any format (numbers, texts, images, sounds, etc.) is represented in a collection of bits; that is, a sequence of 0's and 1's.

Gates

A gate is a tiny device made out of switches designed to perform a very simple task.

2-input AND gate



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

http://www.eng.cam.ac.uk/

The processor and memory

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

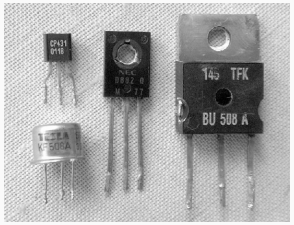
- *A processor* executes computer programs (which are basically algorithms written in a computer language).
- *Memory* stores a large amount of information (in bits). Programs as well as data are stored in memory.

Swiches

A switch is a device that changes the status of a bit from 0 to 1 and vice versa.

In the early days (1940s and 1950s), *vacuum tubes* and *transistors* were used as switches.

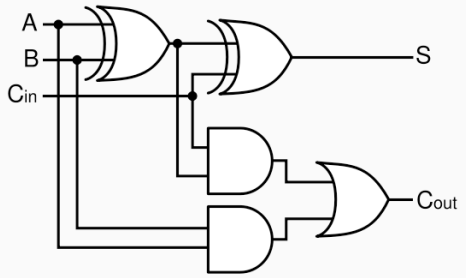
http://www.tablix.org/



http://www.wikipedia.org/

Circuits

A circuit is a tiny device made out of gates designed to perform a simple task.



http://www.wikipedia.org/

John von Neumann
Neumann János Lajos

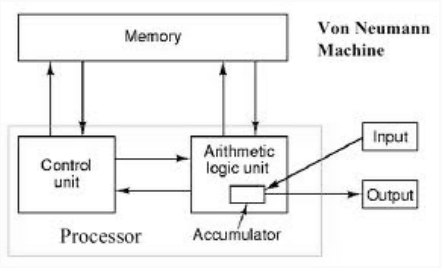
http://www.wikipedia.org/

Software development process

September 12, 2010

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.

Problem analysis

Analyze and formalize the problem to be solved. Determine the *Input/Output (I/O) specification* (which in turn determines what the program will do).

Implementation

- Translate algorithms into program code.
- Test and debug the code.

Software development process

Consists of three basic steps:

- Problem analysis
- Program and Algorithm design
- Implementation

Program and Algorithm design

- Formulate the overall structure of the program and break the program into parts.
- For each part design algorithms that meet the specification.

Software development process

In a nut shell, this consists of the following three basic steps:

- Problem analysis
- Algorithm design
- Implementation

Many skip analysis and design and jump into implementation.

Python's basic constructs

Software development process

In a nut shell, this consists of the following three basic steps:

- Problem analysis
- Algorithm design
- Implementation

Software development process

In a nut shell, this consists of the following three basic steps:

- Problem analysis
- Algorithm design
- Implementation

Many skip analysis and design and jump into implementation — a very bad idea!

Values

Numbers, letters, words, … are collectively called *values*.

Example. The following are all values.

```
1823
3.141592653
'Trinity College'
```


Types

Python has a type command that tells you the type of a given value:

Values

Numbers, letters, words, … are collectively called *values*.

Types

Values are categorized into different *types*: integers, floating-point (real) numbers, strings, …

Each value belongs to a single type (i.e., no value belongs to more than one type).

Types

Python has a type command that tells you the type of a given value:

Example.

```
>>> type(1823)
<type 'int'>
```


Types

Python has a type command that tells you the type of a given value:

Example.

```
>>> type(1823)
<type 'int'>
>>> type(3.141592653)
<type 'float'>
```


Types

Python has a type command that tells you the type of a given value:

Example.

```
>>> type(1823)
```


Types

Python has a type command that tells you the type of a given value:

Example.

```
>>> type(1823)
<type 'int'>
>>> type(3.141592653)
```

Types

Python has a type command that tells you the type of a given value:

Example.

```
>>> type(1823)
<type 'int'>
>>> type(3.141592653)
<type 'float'>
>>> type('Trinity College')
<type 'str'>
```


Variables

A *variable* is a name that refers to a value.

An *assignment* statement creates a variable and assigns a value to it.

Example.

```
>>> n = 1823
```


Types

Python has a type command that tells you the type of a given value:

Example.

```
>>> type(1823)
<type 'int'>
>>> type(3.141592653)
<type 'float'>
>>> type('Trinity College')
```

September 12, 2010

Variables

A *variable* is a name that refers to a value.

An *assignment* statement creates a variable and assigns a value to it.

September 12, 2010

Variables

A *variable* is a name that refers to a value.

An *assignment* statement creates a variable and assigns a value to it.

Example.

```
>>> n = 1823
>>> print n
1823
```


Variables

A *variable* is a name that refers to a value.

An *assignment* statement creates a variable and assigns a value to it.

Example.

```
>>> s = 'Trinity College'
>>> print s
```


Variables

A *variable* is a name that refers to a value.

An *assignment* statement creates a variable and assigns a value to it.

Example.

```
>>> n = 1823
>>> print n
```


Variables

A *variable* is a name that refers to a value.

An *assignment* statement creates a variable and assigns a value to it.

Example.

```
>>> s = 'Trinity College'
```

Variables and types

The *type* of a variable is the type of its value.

Variables and types

The *type* of a variable is the type of its value.

Example.

```
>>> n = 1823
>>> type(n)
```


Variables

A *variable* is a name that refers to a value.

An *assignment* statement creates a variable and assigns a value to it.

Example.

```
>>> s = 'Trinity College'
>>> print s
Trinity College
```


Variables and types

The *type* of a variable is the type of its value.

Example.

```
>>> n = 1823
```

Variables and types

The *type* of a variable is the type of its value.

Example.

```
>>> s = 'Trinity College'
```


Variables and types

The *type* of a variable is the type of its value.

Example.

```
>>> s = 'Trinity College'
>>> type(s)
<type 'str'>
```


Variables and types

The *type* of a variable is the type of its value.

Example.

```
>>> n = 1823
>>> type(n)
<type 'int'>
```


Variables and types

The *type* of a variable is the type of its value.

Example.

```
>>> s = 'Trinity College'
>>> type(s)
```

Variable names

Example.

```
temp = 33
exam3 = 89
my_college = 'Trinity College'
```


Statements

A *statement* is the smallest unit of code that can be executed as a stand-alone instruction.

Variable names

A variable can be named by any word (except for certain reserved words).

A variable name can contain any letter or digit, but it must begin with a letter.

To name a variable using more than one word, it is customary to use _ to connect words.

Variable names

Here are some of the *reserved words* that cannot be used to name variables:

```
and, as, assert, break, class,
continue, def, del, elif, else,
except, exec, finally, for, from,
global, if, import, in, is, lambda,
not, or, pass, print, raise, ...
```

Statements

A *statement* is the smallest unit of code that can be executed as a stand-alone instruction.

Example.

```
10 + 11
```


Expressions

An *expression* is a code fragment that has a value.

Statements

A *statement* is the smallest unit of code that can be executed as a stand-alone instruction.

Example.

```
print 'Hello, World!'
```


Statements

A *statement* is the smallest unit of code that can be executed as a stand-alone instruction.

Example.

```
n = 1823
```

Expressions

An *expression* is a code fragment that has a value.

Example.

`1823 + n` (assuming `n` has a value)

Expressions

An *expression* is a code fragment that has a value.

Non example.

```
n = 1823
```


Expressions

An *expression* is a code fragment that has a value.

Example.

```
1823
```


Expressions

An *expression* is a code fragment that has a value.

Example.

```
'Trinity College'
```

Statement and expressions

An *expression* is usually part of a statement.

Statement and expressions

An *expression* is usually part of a statement.

Example. This statement

```
print 'Trinity College'
```

contains an expression `'Trinity College'`.

Expressions

An *expression* is a code fragment that has a value.

Non example.

```
print 'Trinity College'
```


Statement and expressions

An *expression* is usually part of a statement.

Example. This statement

```
n = 11 * (2 + 3)
```

contains an expression `2 + 3`.