

Outline

- 1 The von Neumann Machine
elements and behavior
executing programs
- 2 Python Programming
developing programs
use as a calculator
converting strings to numbers
- 3 Summary + Assignments

MCS 260 Lecture 3
Introduction to Computer Science
Jan Vershelde, 29 August 2008

Machine Architecture using Python as a calculator

The von
Neumann
Machine

elements and
behavior

executing programs

Python
Programming

developing programs

use as a calculator

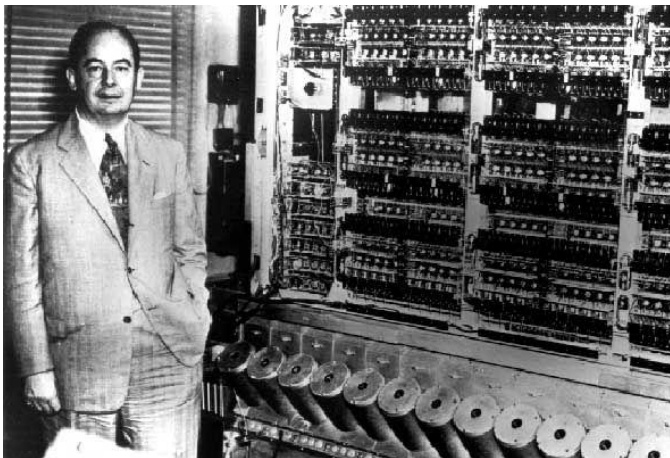
converting strings to
numbers

Summary +
Assignments

- 1 The von Neumann Machine
elements and behavior
executing programs
- 2 Python Programming
developing programs
use as a calculator
converting strings to numbers
- 3 Summary + Assignments

John von Neumann

first computer of the Institute for Advanced Study



[http://www-groups.dcs.st-and.ac.uk/
~history/Mathematicians/Von_Neumann.html](http://www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Von_Neumann.html)

Elements of the Machine

five components

The main components of a computer are

- 1 processor (or CPU) for control and execution;
- 2 memory (or RAM) to store programs and data;
- 3 storage for keeping permanent data;
- 4 peripherals to communicate with the computer;
- 5 system bus for connecting all components.

The processor **fetches** instructions from memory, **decodes** them, and **executes**: *the von Neumann machine*.

The stages of the processor are synchronized by **the system clock**. The frequency of the clock (#ticks per second) gives an upper bound on #arithmetical operations per second the computer can perform.

Elements of the Machine

five components

The main components of a computer are

- 1 processor (or CPU) for control and execution;
- 2 memory (or RAM) to store programs and data;
- 3 storage for keeping permanent data;
- 4 peripherals to communicate with the computer;
- 5 system bus for connecting all components.

The processor **fetches** instructions from memory, **decodes** them, and **executes**: *the von Neumann machine*.

The stages of the processor are synchronized by **the system clock**. The frequency of the clock (#ticks per second) gives an upper bound on #arithmetical operations per second the computer can perform.

Elements of the Machine

five components

The main components of a computer are

- 1 processor (or CPU) for control and execution;
- 2 memory (or RAM) to store programs and data;
- 3 storage for keeping permanent data;
- 4 peripherals to communicate with the computer;
- 5 system bus for connecting all components.

The processor **fetches** instructions from memory, **decodes** them, and **executes**: *the von Neumann machine*.

The stages of the processor are synchronized by **the system clock**. The frequency of the clock (#ticks per second) gives an upper bound on #arithmetical operations per second the computer can perform.

Coding Data and Instructions

binary system, ASCII code, and Unicode

In a computer all information is encoded in binary form.

- The binary number system uses bits $\in \{0, 1\}$.
- Eight bits form a **byte**:
 - 1 with 8 bits we have 2^8 numbers, from 0 to 255;
 - 2 a byte is also called a *word*, fits in 1 memory cell.
- The ASCII (American Standard Code for Information Interchange) code uses 7 bits for 128 characters. Often the code is extended to 8 bits.
- A more extensive code is Unicode:
16 bits, for $2^{16} = 65,536$ different bit patterns, to represent text in languages as Chinese.

Coding Data and Instructions

binary system, ASCII code, and Unicode

In a computer all information is encoded in binary form.

- The binary number system uses bits $\in \{0, 1\}$.
- Eight bits form a **byte**:
 - ① with 8 bits we have 2^8 numbers, from 0 to 255;
 - ② a byte is also called a *word*, fits in 1 memory cell.
- The ASCII (American Standard Code for Information Interchange) code uses 7 bits for 128 characters. Often the code is extended to 8 bits.
- A more extensive code is Unicode: 16 bits, for $2^{16} = 65,536$ different bit patterns, to represent text in languages as Chinese.

Coding Data and Instructions

binary system, ASCII code, and Unicode

In a computer all information is encoded in binary form.

- The binary number system uses bits $\in \{0, 1\}$.
- Eight bits form a **byte**:
 - ① with 8 bits we have 2^8 numbers, from 0 to 255;
 - ② a byte is also called a *word*, fits in 1 memory cell.
- The ASCII (American Standard Code for Information Interchange) code uses 7 bits for 128 characters. Often the code is extended to 8 bits.
- A more extensive code is Unicode:
16 bits, for $2^{16} = 65,536$ different bit patterns,
to represent text in languages as Chinese.

1. Processor

CPU = Central Processing Unit

The processor consists of the following units:

- 1 The **control unit** fetches and decodes instructions. It also sends control signals.
- 2 The **system clock** synchronizes operations.
- 3 The **arithmetic and logic unit (ALU)** performs the operations.

The CPU has several *registers*. A register is a memory cell which can be accessed very quickly.

1. Processor

CPU = Central Processing Unit

The processor consists of the following units:

- 1 The **control unit** fetches and decodes instructions. It also sends control signals.
- 2 The **system clock** synchronizes operations.
- 3 The **arithmetic and logic unit (ALU)** performs the operations.

The CPU has several *registers*. A register is a memory cell which can be accessed very quickly.

1. Processor

CPU = Central Processing Unit

The processor consists of the following units:

- 1 The **control unit** fetches and decodes instructions. It also sends control signals.
- 2 The **system clock** synchronizes operations.
- 3 The **arithmetic and logic unit (ALU)** performs the operations.

The CPU has several *registers*. A register is a memory cell which can be accessed very quickly.

1. Processor

CPU = Central Processing Unit

The processor consists of the following units:

- 1 The **control unit** fetches and decodes instructions. It also sends control signals.
- 2 The **system clock** synchronizes operations.
- 3 The **arithmetic and logic unit (ALU)** performs the operations.

The CPU has several *registers*. A register is a memory cell which can be accessed very quickly.

The Principal CPU Registers

The von Neumann Machine

elements and behavior

executing programs

Python Programming

developing programs

use as a calculator

converting strings to numbers

Summary + Assignments

All data handled by the processor is held in registers.
The principal registers are

- 1 data registers and address registers
- 2 current instruction register
- 3 program counter
- 4 interrupt register
- 5 registers which contain operands
- 6 several working registers

The ALU (Arithmetic and Logic Unit) also uses registers.
One particular register is the status register which carries in its bits information about the results of an operation.

The Principal CPU Registers

The von Neumann Machine

elements and behavior

executing programs

Python Programming

developing programs

use as a calculator

converting strings to numbers

Summary + Assignments

All data handled by the processor is held in registers.
The principal registers are

- 1 data registers and address registers
- 2 current instruction register
- 3 program counter
- 4 interrupt register
- 5 registers which contain operands
- 6 several working registers

The ALU (Arithmetic and Logic Unit) also uses registers.
One particular register is the status register which carries in its bits information about the results of an operation.

The Principal CPU Registers

The von Neumann Machine

elements and behavior

executing programs

Python Programming

developing programs

use as a calculator

converting strings to numbers

Summary + Assignments

All data handled by the processor is held in registers.
The principal registers are

- 1 data registers and address registers
- 2 current instruction register
- 3 program counter
- 4 interrupt register
- 5 registers which contain operands
- 6 several working registers

The ALU (Arithmetic and Logic Unit) also uses registers.
One particular register is the status register which carries in its bits information about the results of an operation.

2. Main Memory

RAM and ROM

- RAM = Random Access Memory, available to user;
ROM = Read Only Memory, used to store microprograms delivered by the manufacturer.
- The main memory is a sequence of memory cells.
The size of one memory cell is one or several bytes.
- Every memory cell has an address. Memory addressing happens via an address register.
An address register of k bits reaches 2^k cells.
A 32-bit address register means that 4Gb of RAM can be addressed. New 64-bit memory architectures can address 16 exabytes.
- Between the registers and main memory: **cache**.

2. Main Memory

RAM and ROM

- RAM = Random Access Memory, available to user;
ROM = Read Only Memory, used to store microprograms delivered by the manufacturer.
- The main memory is a sequence of memory cells. The size of one memory cell is one or several bytes.
- Every memory cell has an address. Memory addressing happens via an address register. An address register of k bits reaches 2^k cells. A 32-bit address register means that 4Gb of RAM can be addressed. New 64-bit memory architectures can address 16 exabytes.
- Between the registers and main memory: **cache**.

2. Main Memory

RAM and ROM

- RAM = Random Access Memory, available to user;
ROM = Read Only Memory, used to store microprograms delivered by the manufacturer.
- The main memory is a sequence of memory cells. The size of one memory cell is one or several bytes.
- Every memory cell has an address. Memory addressing happens via an address register. An address register of k bits reaches 2^k cells. A 32-bit address register means that 4Gb of RAM can be addressed. New 64-bit memory architectures can address 16 exabytes.
- Between the registers and main memory: **cache**.

2. Main Memory

RAM and ROM

- RAM = Random Access Memory, available to user;
ROM = Read Only Memory, used to store microprograms delivered by the manufacturer.
- The main memory is a sequence of memory cells. The size of one memory cell is one or several bytes.
- Every memory cell has an address. Memory addressing happens via an address register. An address register of k bits reaches 2^k cells. A 32-bit address register means that 4Gb of RAM can be addressed. New 64-bit memory architectures can address 16 exabytes.
- Between the registers and main memory: **cache**.

3. Mass Storage

to store permanently

Older technology includes tapes and floppy disks.

We distinguish between removable and hard drives:

- 1 the hard drive, the disk inside the computer;
- 2 USB memory keys act as hard drives, but portable;
- 3 CD-ROM = Compact Disc Read Only Memory
DVD = Digital Versatile Disc

The organization of information is often as a tree

- a file stores a program or data;
- a directory contains several files.

3. Mass Storage

to store permanently

Older technology includes tapes and floppy disks.

We distinguish between removable and hard drives:

- 1 the hard drive, the disk inside the computer;
- 2 USB memory keys act as hard drives, but portable;
- 3 CD-ROM = Compact Disc Read Only Memory
DVD = Digital Versatile Disc

The organization of information is often as a tree

- a file stores a program or data;
- a directory contains several files.

4. Input/Output Interfaces

to connect to peripherals

Input/Output interfaces form the circuitry to connect the computer with the peripherals.

An interface typically contains the following elements:

- 1 a peripheral data register
- 2 a peripheral command register
- 3 status information about the peripheral

Some of these interfaces have grown very intelligent, i.e.: video cards have become complex and powerful. The success of the gaming industry led to the cell processor.

4. Input/Output Interfaces

to connect to peripherals

Input/Output interfaces form the circuitry to connect the computer with the peripherals.

An interface typically contains the following elements:

- 1 a peripheral data register
- 2 a peripheral command register
- 3 status information about the peripheral

Some of these interfaces have grown very intelligent, i.e.: video cards have become complex and powerful. The success of the gaming industry led to the cell processor.

4. Input/Output Interfaces

to connect to peripherals

Input/Output interfaces form the circuitry to connect the computer with the peripherals.

An interface typically contains the following elements:

- 1 a peripheral data register
- 2 a peripheral command register
- 3 status information about the peripheral

Some of these interfaces have grown very intelligent, i.e.: video cards have become complex and powerful. The success of the gaming industry led to the cell processor.

5. System Bus

for connecting components

The processor activates other units via the system bus.

Data is divided in three categories:

data bus transfers data from memory into data register
and vice versa;

address bus transmits contents of address register to main
memory;

control bus transfers instructions to be executed.

5. System Bus

for connecting components

The processor activates other units via the system bus.

Data is divided in three categories:

data bus transfers data from memory into data register and vice versa;

address bus transmits contents of address register to main memory;

control bus transfers instructions to be executed.

5. System Bus

for connecting components

The processor activates other units via the system bus.

Data is divided in three categories:

data bus transfers data from memory into data register and vice versa;

address bus transmits contents of address register to main memory;

control bus transfers instructions to be executed.

Machine Architecture using Python as a calculator

The von Neumann Machine

elements and
behavior

executing programs

Python Programming

developing programs

use as a calculator

converting strings to
numbers

Summary + Assignments

1 The von Neumann Machine

elements and behavior

executing programs

2 Python Programming

developing programs

use as a calculator

converting strings to numbers

3 Summary + Assignments

Executing Programs

imagine what your program does

Consider the python command `print 'hello world'`.

The following tasks need to happen:

- 1 fetch the instruction
- 2 decode the instruction as a print
- 3 get the argument of the print
- 4 pass the command to the screen device

Each of this four steps involves many more steps...

Executing Programs

imagine what your program does

Consider the python command `print 'hello world'`.

The following tasks need to happen:

- 1 fetch the instruction
- 2 decode the instruction as a print
- 3 get the argument of the print
- 4 pass the command to the screen device

Each of this four steps involves many more steps...

Executing Programs

imagine what your program does

Consider the python command `print 'hello world'`.

The following tasks need to happen:

- 1 fetch the instruction
- 2 decode the instruction as a print
- 3 get the argument of the print
- 4 pass the command to the screen device

Each of this four steps involves many more steps...

Executing Programs

imagine what your program does

Consider the python command `print 'hello world'`.

The following tasks need to happen:

- 1 fetch the instruction
- 2 decode the instruction as a print
- 3 get the argument of the print
- 4 pass the command to the screen device

Each of this four steps involves many more steps...

Executing Programs

imagine what your program does

Consider the python command `print 'hello world'`.

The following tasks need to happen:

- 1 fetch the instruction
- 2 decode the instruction as a print
- 3 get the argument of the print
- 4 pass the command to the screen device

Each of this four steps involves many more steps...

Executing Programs

imagine what your program does

Consider the python command `print 'hello world'`.

The following tasks need to happen:

- 1 fetch the instruction
- 2 decode the instruction as a print
- 3 get the argument of the print
- 4 pass the command to the screen device

Each of this four steps involves many more steps...

Assembly Language

Machine instructions are sequences of bits.

Some terminology:

- An *assembly language* is a mnemonic system for representing machine instructions.
- An *assembler* is a program to translate assembly language into machine instructions.

Writing assembly language has some advantages:

- 1 develop good understanding of how it really works
- 2 the resulting code can be very efficient

Drawbacks of programming in assembly language:

- 1 requires a very disciplined style of coding
- 2 assembler language depends on the architecture and the code is often not portable

Assembly Language

The von
Neumann
Machine

elements and
behavior
executing programs

Python
Programming

developing programs
use as a calculator
converting strings to
numbers

Summary +
Assignments

Machine instructions are sequences of bits.

Some terminology:

- An *assembly language* is a mnemonic system for representing machine instructions.
- An *assembler* is a program to translate assembly language into machine instructions.

Writing assembly language has some advantages:

- 1 develop good understanding of how it really works
- 2 the resulting code can be very efficient

Drawbacks of programming in assembly language:

- 1 requires a very disciplined style of coding
- 2 assembler language depends on the architecture and the code is often not portable

Machine Architecture using Python as a calculator

The von Neumann Machine

elements and
behavior
executing programs

Python Programming

developing programs
use as a calculator
converting strings to
numbers

Summary + Assignments

- 1 The von Neumann Machine
elements and behavior
executing programs
- 2 Python Programming
developing programs
use as a calculator
converting strings to numbers
- 3 Summary + Assignments

Developing Python programs

Use python interactively, or run `m.py` as `python m.py`.

Python is installed on icarus.cc.uic.edu.

As Python is interpreted, use incremental development:

- 1 first try the commands in an interactive session
- 2 save the working commands in a file (e.g. as `w.py`)
- 3 test the program, run as `python w.py`
- 4 if the run fails, retry the commands interactively
- 5 after success, continue step 1 with new commands

Save intermediate versions of [working](#) programs.

Developing Python programs

Use python interactively, or run `m.py` as `python m.py`.

Python is installed on icarus.cc.uic.edu.

As Python is interpreted, use incremental development:

- 1 first try the commands in an interactive session
- 2 save the working commands in a file (e.g. as `w.py`)
- 3 test the program, run as `python w.py`
- 4 if the run fails, retry the commands interactively
- 5 after success, continue step 1 with new commands

Save intermediate versions of **working** programs.

Developing Python programs

The von
Neumann
Machine

elements and
behavior
executing programs

Python
Programming

developing programs
use as a calculator
converting strings to
numbers

Summary +
Assignments

Use python interactively, or run `m.py` as `python m.py`.

Python is installed on icarus.cc.uic.edu.

As Python is interpreted, use incremental development:

- 1 first try the commands in an interactive session
- 2 save the working commands in a file (e.g. as `w.py`)
- 3 test the program, run as `python w.py`
- 4 if the run fails, retry the commands interactively
- 5 after success, continue step 1 with new commands

Save intermediate versions of [working](#) programs.

Developing Python programs

The von
Neumann
Machine

elements and
behavior
executing programs

Python
Programming

developing programs
use as a calculator
converting strings to
numbers

Summary +
Assignments

Use python interactively, or run `m.py` as `python m.py`.

Python is installed on icarus.cc.uic.edu.

As Python is interpreted, use incremental development:

- 1 first try the commands in an interactive session
- 2 save the working commands in a file (e.g. as `w.py`)
- 3 test the program, run as `python w.py`
- 4 if the run fails, retry the commands interactively
- 5 after success, continue step 1 with new commands

Save intermediate versions of [working](#) programs.

Developing Python programs

The von
Neumann
Machine

elements and
behavior
executing programs

Python
Programming

developing programs
use as a calculator
converting strings to
numbers

Summary +
Assignments

Use python interactively, or run `m.py` as `python m.py`.

Python is installed on icarus.cc.uic.edu.

As Python is interpreted, use incremental development:

- 1 first try the commands in an interactive session
- 2 save the working commands in a file (e.g. as `w.py`)
- 3 test the program, run as `python w.py`
- 4 if the run fails, retry the commands interactively
- 5 after success, continue step 1 with new commands

Save intermediate versions of [working](#) programs.

Developing Python programs

The von
Neumann
Machine

elements and
behavior
executing programs

Python
Programming

developing programs
use as a calculator
converting strings to
numbers

Summary +
Assignments

Use python interactively, or run `m.py` as `python m.py`.

Python is installed on icarus.cc.uic.edu.

As Python is interpreted, use incremental development:

- 1 first try the commands in an interactive session
- 2 save the working commands in a file (e.g. as `w.py`)
- 3 test the program, run as `python w.py`
- 4 if the run fails, retry the commands interactively
- 5 after success, continue step 1 with new commands

Save intermediate versions of [working](#) programs.

Developing Python programs

The von
Neumann
Machine

elements and
behavior
executing programs

Python
Programming

developing programs
use as a calculator
converting strings to
numbers

Summary +
Assignments

Use python interactively, or run `m.py` as `python m.py`.

Python is installed on icarus.cc.uic.edu.

As Python is interpreted, use incremental development:

- 1 first try the commands in an interactive session
- 2 save the working commands in a file (e.g. as `w.py`)
- 3 test the program, run as `python w.py`
- 4 if the run fails, retry the commands interactively
- 5 after success, continue step 1 with new commands

Save intermediate versions of [working](#) programs.

Developing Python programs

The von
Neumann
Machine

elements and
behavior
executing programs

Python
Programming

developing programs
use as a calculator
converting strings to
numbers

Summary +
Assignments

Use python interactively, or run `m.py` as `python m.py`.

Python is installed on icarus.cc.uic.edu.

As Python is interpreted, use incremental development:

- 1 first try the commands in an interactive session
- 2 save the working commands in a file (e.g. as `w.py`)
- 3 test the program, run as `python w.py`
- 4 if the run fails, retry the commands interactively
- 5 after success, continue step 1 with new commands

Save intermediate versions of **working** programs.

Using icarus

SUN server for UIC students

Connecting to icarus from a Terminal Window on a Mac in SEL 2263 with **ssh** (*secure shell*):

```
prompt$ ssh icarus.cc.uic.edu -l netid
```

Normally the same `netid` and password with which you connected in the lab should work. If not, send an email to `consult@uic.edu` to request an account on icarus.

Saving files on icarus with **scp** (*secure copy*):

```
$ scp hello.py netid@icarus.cc.uic.edu:~
```

Retrieving files from icarus:

```
$ scp netid@icarus.cc.uic.edu:~/hello.py .
```

Using icarus

SUN server for UIC students

Connecting to icarus from a Terminal Window on a Mac in SEL 2263 with **ssh** (*secure shell*):

```
prompt$ ssh icarus.cc.uic.edu -l netid
```

Normally the same `netid` and password with which you connected in the lab should work. If not, send an email to `consult@uic.edu` to request an account on icarus.

Saving files on icarus with **scp** (*secure copy*):

```
$ scp hello.py netid@icarus.cc.uic.edu:~
```

Retrieving files from icarus:

```
$ scp netid@icarus.cc.uic.edu:~/hello.py .
```

Machine Architecture using Python as a calculator

The von Neumann Machine

elements and
behavior
executing programs

Python Programming

developing programs
use as a calculator
converting strings to
numbers

Summary + Assignments

- 1 The von Neumann Machine
elements and behavior
executing programs
- 2 Python Programming
developing programs
use as a calculator
converting strings to numbers
- 3 Summary + Assignments

use Python as calculator

- Type expressions at the python prompt
operations: +, -, *, /, and exponentiation **.

- Long integers and floats

```
>>> (2**10)**10
1267650600228229401496703205376L
```

```
>>> (2.0**10)**10
1.2676506002282294e+30
```

Floats are limited in size, long integers are not.

- store results with the assignment

```
>>> x = 2**10
>>> x**10
```

useful to control the order of calculations

use Python as calculator

- Type expressions at the python prompt
operations: +, -, *, /, and exponentiation **.

- Long integers and floats

```
>>> (2**10)**10
1267650600228229401496703205376L
```

```
>>> (2.0**10)**10
1.2676506002282294e+30
```

Floats are limited in size, long integers are not.

- store results with the assignment

```
>>> x = 2**10
>>> x**10
```

useful to control the order of calculations

use Python as calculator

The von Neumann Machine

elements and behavior
executing programs

Python Programming

developing programs
use as a calculator
converting strings to numbers

Summary + Assignments

- Type expressions at the python prompt
operations: +, -, *, /, and exponentiation **.

- Long integers and floats

```
>>> (2**10)**10
1267650600228229401496703205376L
```

```
>>> (2.0**10)**10
1.2676506002282294e+30
```

Floats are limited in size, long integers are not.

- store results with the assignment

```
>>> x = 2**10
>>> x**10
```

useful to control the order of calculations

ASCII Codes in Python

The function `ord` returns the ASCII code of a character:

```
>>> ord('a')
97
>>> ord('1')
49
>>> ord('?')
63
>>> 'a' > '?'
True
```

The function `chr` returns the character with given order:

```
>>> chr(98)
'b'
```

ASCII Codes in Python

The function `ord` returns the ASCII code of a character:

```
>>> ord('a')
97
>>> ord('1')
49
>>> ord('?')
63
>>> 'a' > '?'
True
```

The function `chr` returns the character with given order:

```
>>> chr(98)
'b'
```

Machine Architecture using Python as a calculator

The von Neumann Machine

elements and
behavior
executing programs

Python Programming

developing programs
use as a calculator
converting strings to
numbers

Summary + Assignments

- 1 The von Neumann Machine
elements and behavior
executing programs
- 2 Python Programming
developing programs
use as a calculator
converting strings to numbers
- 3 Summary + Assignments

Converting Strings to Numbers

after `raw_input()`

The von
Neumann
Machine

elements and
behavior
executing programs

Python
Programming

developing programs
use as a calculator
converting strings to
numbers

Summary +
Assignments

The result of `raw_input()` is a string.

If this string represents a number, before we can calculate with it, we must convert it to a number type, e.g.: `float`.

```
>>> n = raw_input('give a number : ');
give a number : 3.4
>>> float(n)
3.3999999999999999
>>> float(n) - 3.4
0.0
```

Converting Strings to Numbers

after `raw_input()`

The von
Neumann
Machine

elements and
behavior
executing programs

Python
Programming

developing programs
use as a calculator
converting strings to
numbers

Summary +
Assignments

The result of `raw_input()` is a string.

If this string represents a number, before we can calculate with it, we must convert it to a number type, e.g.: `float`.

```
>>> n = raw_input('give a number : ');
give a number : 3.4
>>> float(n)
3.3999999999999999
>>> float(n) - 3.4
0.0
```

Converting Numbers to Strings

for nice output formats

Sometimes only few decimal places of a floating-point number are needed for display.

Use of the % Operator:

```
>>> x = 1.238
>>> sx = '%.2f' % x
>>> print sx
1.24
```

Types of the variables `x` and `sx` are different:

```
>>> type(x)
<type 'float'>
>>> type(sx)
<type 'str'>
```

Converting Numbers to Strings

for nice output formats

Sometimes only few decimal places of a floating-point number are needed for display.

Use of the % Operator:

```
>>> x = 1.238
>>> sx = '%.2f' % x
>>> print sx
1.24
```

Types of the variables `x` and `sx` are different:

```
>>> type(x)
<type 'float'>
>>> type(sx)
<type 'str'>
```

Converting Numbers to Strings

for nice output formats

Sometimes only few decimal places of a floating-point number are needed for display.

Use of the % Operator:

```
>>> x = 1.238
>>> sx = '%.2f' % x
>>> print sx
1.24
```

Types of the variables `x` and `sx` are different:

```
>>> type(x)
<type 'float'>
>>> type(sx)
<type 'str'>
```

Summary + Assignments

The von
Neumann
Machine

elements and
behavior
executing programs

Python
Programming

developing programs
use as a calculator
converting strings to
numbers

Summary +
Assignments

Background material for this lecture:

- chapter 2 of *Computer Science. An Overview*; and
- topics of chapter 4 of *Python Power!*

Assignments:

- 1 Suppose two numbers are in main memory. Describe the fetch-decode-execute cycle to compute their sum and place the sum in main memory.
- 2 Compute $1/3$ and $1.0/3$ in Python. Explain why the answers are so different.
- 3 Represent the string `hello` as a string of bits, replacing each character by its ASCII code. Give the hexadecimal representation of the bit string.

Answers to selected exercises will be collected around the third week, *after* the deadline for project one.