Outline

1. Digital Systems
   transistors
   logic gates

2. Intrinsic Operations
   on numbers
   on strings

3. Dictionaries and Conditionals
   writing numbers in words
   algorithm → flowchart → code

4. Summary + Assignments

MCS 260 Lecture 9
Introduction to Computer Science
Jan Verschelde, 15 September 2008
A computer is a synchronous binary digital system.

digital: all information is discrete (not continuous)

binary: only zero and one are used
  a binary digit is a bit

synchronous: functioning is ruled by the system clock

Basic elements to represent bits are switches that can be open (1) or closed (0).

Transistors are electronic circuits to represent bits.
A computer is a synchronous binary digital system.

**digital:** all information is discrete (not continuous)

**binary:** only zero and one are used

A binary digit is a bit

**synchronous:** functioning is ruled by the system clock

Basic elements to represent bits are switches that can be open (1) or closed (0).

Transistors are electronic circuits to represent bits.
A computer is a synchronous binary digital system.

**digital**: all information is discrete (not continuous)

**binary**: only zero and one are used

a **binary digit** is a bit

**synchronous**: functioning is ruled by the system clock

Basic elements to represent bits are switches that can be open (1) or closed (0).

Transistors are electronic circuits to represent bits.
Digital Systems

introduction to electronic circuits

A computer is a synchronous binary digital system.

digital: all information is discrete (not continuous)

binary: only zero and one are used
  a binary digit is a bit

synchronous: functioning is ruled by the system clock

Basic elements to represent bits are switches that can be open (1) or closed (0).

Transistors are electronic circuits to represent bits.
A computer is a synchronous binary digital system.

digital: all information is discrete (not continuous)

binary: only zero and one are used
a binary digit is a bit

synchronous: functioning is ruled by the system clock

Basic elements to represent bits are switches that can be open (1) or closed (0).

Transistors are electronic circuits to represent bits.
A computer is a synchronous binary digital system.

- **digital**: all information is discrete (not continuous)
- **binary**: only zero and one are used
- a **binary digit** is a bit
- **synchronous**: functioning is ruled by the system clock

Basic elements to represent bits are switches that can be open (1) or closed (0).

Transistors are electronic circuits to represent bits.
transistors and gates
intrinsic operations

1. Digital Systems
   transistors
   logic gates

2. Intrinsic Operations
   on numbers
   on strings

3. Dictionaries and Conditionals
   writing numbers in words
   algorithm → flowchart → code

4. Summary + Assignments
Transistors have three connections to the outside:

1. **base**: input voltage
2. **collector**: output voltage
3. **emitter**: to ground

High Voltage: 1
Low Voltage: 0
Transistors have three connections to the outside:

1. base: input voltage
2. collector: output voltage
3. emitter: to ground

High Voltage: 1
Low Voltage: 0
Transistors have three connections to the outside:

1. **base**: input voltage
2. **collector**: output voltage
3. **emitter**: to ground

High Voltage: 1
Low Voltage: 0
transistors and gates
intrinsic operations

1. Digital Systems
   - transistors
   - logic gates

2. Intrinsic Operations
   - on numbers
   - on strings

3. Dictionaries and Conditionals
   - writing numbers in words
   - algorithm → flowchart → code

4. Summary + Assignments
Logic gates are circuits that correspond to logic operators.

Representations of NOT, AND, OR:

\[
\begin{align*}
\text{NOT} & : \quad x \text{ NAND } y = \text{ NOT } (x \text{ AND } y) \\
\text{AND} & : \quad x \text{ NOR } y = \text{ NOT } (x \text{ OR } y)
\end{align*}
\]
Logic gates are circuits that correspond to logic operators.

Representations of NOT, AND, OR:

NOT

\[ x \text{ NAND } y = \text{ NOT} \ (x \text{ AND } y) \]

AND

\[ x \text{ NOR } y = \text{ NOT} \ (x \text{ OR } y) \]
Logic Gates

implement logic operators

Logic gates are circuits that correspond to logic operators.

Representations of NOT, AND, OR:

\[
\begin{align*}
\text{NOT} & : & \Box & & \text{AND} & : & \bigotimes & & \text{OR} & : & \bigotimes \\
& & & & \text{NAND} & : & \bigotimes & & \text{NOR} & : & \bigotimes \\
\end{align*}
\]

\[
\begin{align*}
x \text{ NAND } y & = \text{ NOT } (x \text{ AND } y) \\
x \text{ NOR } y & = \text{ NOT } (x \text{ OR } y)
\end{align*}
\]
A NOT Gate

as realized by a transistor

Input Voltage $V_{in}$

$V_{in} = \text{low}$

$\Rightarrow$ switch is open

$\Rightarrow V_{out} = + V_{\infty}$

$V_{in} = \text{high}$

$\Rightarrow$ switch is closed

$\Rightarrow V_{out} = \text{low}$

A NOT gate converts a low input voltage to high and a high input voltage to low.
A NOT Gate
as realized by a transistor

Input Voltage $V_{\text{in}}$

$V_{\text{in}} = \text{low}$
$\Rightarrow$ switch is open
$\Rightarrow V_{\text{out}} = +V_{\infty}$

$V_{\text{in}} = \text{high}$
$\Rightarrow$ switch is closed
$\Rightarrow V_{\text{out}} = \text{low}$

A NOT gate converts a low input voltage to high
and a high input voltage to low.
A NOT Gate

as realized by a transistor

Input Voltage $V_{in}$

$V_{in} = \text{low}$

$\Rightarrow$ switch is open

$\Rightarrow V_{out} = + V_\infty$

$V_{in} = \text{high}$

$\Rightarrow$ switch is closed

$\Rightarrow V_{out} = \text{low}$

A NOT gate converts a low input voltage to high and a high input voltage to low.
A NOT Gate as realized by a transistor

**Input Voltage** $V_{\text{in}}$

- $V_{\text{in}} = \text{low} \implies \text{switch is open}$
- $V_{\text{in}} = \text{high} \implies \text{switch is closed}$

Thus, $V_{\text{out}}$:

- $V_{\text{out}} = +V_{\infty}$ for low input voltage
- $V_{\text{out}} = \text{low}$ for high input voltage

A NOT gate converts a low input voltage to high and a high input voltage to low.
A NOT Gate
as realized by a transistor

\[ V_{\text{in}} = \text{low} \]
\[ \Rightarrow \text{switch is open} \]
\[ \Rightarrow V_{\text{out}} = +V_{\infty} \]

\[ V_{\text{in}} = \text{high} \]
\[ \Rightarrow \text{switch is closed} \]
\[ \Rightarrow V_{\text{out}} = \text{low} \]

A NOT gate converts a low input voltage to high
and a high input voltage to low.
A NOT Gate

as realized by a transistor

Input Voltage $V_{in}$

$V_{in} = \text{low}$

$\Rightarrow\ 	ext{switch is open}$

$\Rightarrow\ V_{out} = +V_{\infty}$

$V_{in} = \text{high}$

$\Rightarrow\ 	ext{switch is closed}$

$\Rightarrow\ V_{out} = \text{low}$

A NOT gate converts a low input voltage to high and a high input voltage to low.
A NAND Gate

two transistors in series

Input voltages $V_1$ and $V_2$

If either $V_1$ or $V_2$ is low:
$\Rightarrow$ switch is open
$\Rightarrow V_{out} = +V_\infty$

If both $V_1$ and $V_2$ are high:
$\Rightarrow$ switch is closed
$\Rightarrow V_{out} = \text{low}$
A NAND Gate
two transistors in series

Input voltages $V_1$ and $V_2$

If either $V_1$ or $V_2$ is low:

$\Rightarrow$ switch is open

$\Rightarrow V_{out} = +V_\infty$

If both $V_1$ and $V_2$ are high:

$\Rightarrow$ switch is closed

$\Rightarrow V_{out} = \text{low}$
A NAND Gate

two transistors in series

Input voltages $V_1$ and $V_2$

If either $V_1$ or $V_2$ is low:

$\Rightarrow$ switch is open

$\Rightarrow V_{out} = +V_\infty$

If both $V_1$ and $V_2$ are high:

$\Rightarrow$ switch is closed

$\Rightarrow V_{out} = \text{low}$
A NAND Gate

two transistors in series

Input voltages $V_1$ and $V_2$

If either $V_1$ or $V_2$ is low:
$\Rightarrow$ switch is open
$\Rightarrow V_{out} = +V_\infty$

If both $V_1$ and $V_2$ are high:
$\Rightarrow$ switch is closed
$\Rightarrow V_{out} = \text{low}$
A NAND Gate

two transistors in series

Input voltages $V_1$ and $V_2$

If either $V_1$ or $V_2$ is low:
  $\Rightarrow$ switch is open
  $\Rightarrow V_{out} = +V_\infty$

If both $V_1$ and $V_2$ are high:
  $\Rightarrow$ switch is closed
  $\Rightarrow V_{out} = \text{low}$
**A NAND Gate**

two transistors in series

Input voltages $V_1$ and $V_2$

If either $V_1$ or $V_2$ is low:

$\Rightarrow$ switch is open

$\Rightarrow V_{out} = +V_{\infty}$

If both $V_1$ and $V_2$ are high:

$\Rightarrow$ switch is closed

$\Rightarrow V_{out} = \text{low}$
A NAND Gate

two transistors in series

Input voltages $V_1$ and $V_2$

If either $V_1$ or $V_2$ is low:
  $\Rightarrow$ switch is open
  $\Rightarrow V_{out} = +V_\infty$

If both $V_1$ and $V_2$ are high:
  $\Rightarrow$ switch is closed
  $\Rightarrow V_{out} = \text{low}$
A NAND Gate
two transistors in series

Input voltages $V_1$ and $V_2$

If either $V_1$ or $V_2$ is low:
\[ \Rightarrow \text{switch is open} \]
\[ \Rightarrow V_{out} = +V_{\infty} \]

If both $V_1$ and $V_2$ are high:
\[ \Rightarrow \text{switch is closed} \]
\[ \Rightarrow V_{out} = \text{low} \]
A NOR Gate

two transistors in parallel

Input voltages $V_1$ and $V_2$

if either $V_1$ or $V_2$ is high $\Rightarrow$ closed switch $\Rightarrow$ $V_{out} =$ low;
if both $V_1$ or $V_2$ are low $\Rightarrow$ open switch $\Rightarrow$ $V_{out} = +V_\infty$. 

\[ V_{out} \]
A NOR Gate

two transistors in parallel

Input voltages $V_1$ and $V_2$
if either $V_1$ or $V_2$ is high $\Rightarrow$ closed switch $\Rightarrow$ $V_{out} = \text{low}$;
if both $V_1$ or $V_2$ are low $\Rightarrow$ open switch $\Rightarrow$ $V_{out} = +V_\infty$. 
A NOR Gate

two transistors in parallel

Input voltages \( V_1 \) and \( V_2 \)
if either \( V_1 \) or \( V_2 \) is high \( \Rightarrow \) closed switch \( \Rightarrow \) \( V_{\text{out}} = \text{low} \);
if both \( V_1 \) or \( V_2 \) are low \( \Rightarrow \) open switch \( \Rightarrow \) \( V_{\text{out}} = +V_\infty \).
A NOR Gate

two transistors in parallel

Input voltages $V_1$ and $V_2$
if either $V_1$ or $V_2$ is high $\Rightarrow$ closed switch $\Rightarrow$ $V_{out} = \text{low}$;
if both $V_1$ or $V_2$ are low $\Rightarrow$ open switch $\Rightarrow$ $V_{out} = +V_\infty$. 
A NOR Gate

two transistors in parallel

Input voltages $V_1$ and $V_2$
if either $V_1$ or $V_2$ is high $\Rightarrow$ closed switch $\Rightarrow$ $V_{out} = \text{low}$;
if both $V_1$ or $V_2$ are low $\Rightarrow$ open switch $\Rightarrow$ $V_{out} = +V_{\infty}$.
Input voltages $V_1$ and $V_2$
if either $V_1$ or $V_2$ is high ⇒ closed switch ⇒ $V_{out} = \text{low}$;
if both $V_1$ or $V_2$ are low ⇒ open switch ⇒ $V_{out} = +V_{\infty}$. 
Intrinsic Operations

Intrinsic operations are those operations that belong to the standard library.

For every variable $x$, the function

$$\text{id}(x)$$ returns the address of $x$,

$$\text{type}(x)$$ returns the type of $x$.

Python has dynamic typing and garbage collection.
Intrinsic Operations

Intrinsic operations are those operations that belong to the standard library.

For every variable \( x \), the function

\[
id(x) \text{ returns the address of } x,
\]

\[
type(x) \text{ returns the type of } x.
\]

Python has dynamic typing and garbage collection.
Intrinsic Operations

Intrinsic operations are those operations that belong to the standard library.

For every variable $x$, the function

\[ \text{id}(x) \] returns the address of $x$,

\[ \text{type}(x) \] returns the type of $x$.

Python has dynamic typing and garbage collection.
Intrinsic Operations

Intrinsic operations are those operations that belong to the standard library.

For every variable $x$, the function

$$\text{id}(x) \text{ returns the address of } x,$$

$$\text{type}(x) \text{ returns the type of } x.$$

Python has dynamic typing and garbage collection.
transistors and gates
intrinsic operations

1. Digital Systems
   transistors
   logic gates

2. Intrinsic Operations
   on numbers
   on strings

3. Dictionaries and Conditionals
   writing numbers in words
   algorithm $\rightarrow$ flowchart $\rightarrow$ code

4. Summary + Assignments
Conversions for Numbers

<table>
<thead>
<tr>
<th>function</th>
<th>converts ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>int()</td>
<td>string or number to integer</td>
</tr>
<tr>
<td>long()</td>
<td>string or number to long integer</td>
</tr>
<tr>
<td>float()</td>
<td>string or number to float</td>
</tr>
<tr>
<td>complex()</td>
<td>string or number to complex number</td>
</tr>
</tbody>
</table>

Examples: \( j = \sqrt{-1}, \) the imaginary unit

```python
>>> complex(1)
(1+0j)
>>> complex('89j')
89j
>>> _ + complex(3,4)
(3+93j)
```
Conversions for Numbers

<table>
<thead>
<tr>
<th>function</th>
<th>converts ...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int()</code></td>
<td>string or number to integer</td>
</tr>
<tr>
<td><code>long()</code></td>
<td>string or number to long integer</td>
</tr>
<tr>
<td><code>float()</code></td>
<td>string or number to float</td>
</tr>
<tr>
<td><code>complex()</code></td>
<td>string or number to complex number</td>
</tr>
</tbody>
</table>

Examples:

```python
>>> complex(1)
(1+0j)
>>> complex('89j')
89j
>>> _ + complex(3,4)
(3+93j)
```

(j = \sqrt{-1}, the imaginary unit)
transistors and gates
intrinsic operations

1. Digital Systems
   transistors
   logic gates

2. Intrinsic Operations
   on numbers
   on strings

3. Dictionaries and Conditionals
   writing numbers in words
   algorithm → flowchart → code

4. Summary + Assignments
Intrinsic Operations for Strings

Converting numbers to strings:

```python
>>> str(12*3)
'36'
```

Observe the use of right quotes:

```python
>>> str('12*3')
'12*3'
```

Right quotes prevent the evaluation.

With left quotes, as in `str('12*3')`, the expression `12*3` is evaluated first before the conversion.
Intrinsic Operations for Strings

Converting numbers to strings:

```python
>>> str(12*3)
'36'
```

Observe the use of right quotes:

```python
>>> str('12*3')
'12*3'
```

Right quotes prevent the evaluation.

With left quotes, as in `str('12*3')`, the expression `12*3` is evaluated first before the conversion.
Intrinsic Operations for Strings

Converting numbers to strings:
```
>>> str(12*3)
'36'
```

Observe the use of right quotes:
```
>>> str('12*3')
'12*3'
```
```
>>> '12*3'
'12*3'
```

Right quotes prevent the evaluation.
With left quotes, as in `str('12*3')`, the expression `12*3` is evaluated first before the conversion.
Representations of Objects

The function `repr()` is very similar to `str()`. `repr()` returns the representation of an object.

For floating-point numbers, the output of `repr()` and `str()` may differ:

```python
>>> repr(13.3)
'13.300000000000001'
>>> str(13.3)
'13.3'
```

Internal representations of numbers are binary, `str()` and `repr()` use a different number of digits for the decimal expansion of a float, respectively 12 and 17.
Representations of Objects

The function `repr()` is very similar to `str()`. `repr()` returns the representation of an object.

For floating-point numbers, the output of `repr()` and `str()` may differ:

```python
>>> repr(13.3)
'13.300000000000001'
>>> str(13.3)
'13.3'
```

Internal representations of numbers are binary, `str()` and `repr()` use a different number of digits for the decimal expansion of a float, respectively 12 and 17.
Tests on Strings

built-in methods

For a string $s$, we have the methods

<table>
<thead>
<tr>
<th>method</th>
<th>returns True if ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$.islower()</td>
<td>$s$ in lower case</td>
</tr>
<tr>
<td>$s$.isupper()</td>
<td>$s$ in upper case</td>
</tr>
<tr>
<td>$s$.istitle()</td>
<td>$s$ in title form</td>
</tr>
<tr>
<td>$s$.isdigit()</td>
<td>$s$ contains only digits</td>
</tr>
<tr>
<td>$s$.isalpha()</td>
<td>$s$ contains only letters</td>
</tr>
<tr>
<td>$s$.isalnum()</td>
<td>$s$ contains only letters and digits</td>
</tr>
</tbody>
</table>

Examples:

$x = 'hello' \Rightarrow x$.islower() is True
$\Rightarrow x$.isalpha() is True
$\Rightarrow x$.isalnum() is True
Tests on Strings
built-in methods

For a string $s$, we have the methods

<table>
<thead>
<tr>
<th>method</th>
<th>returns True if ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$.islower()</td>
<td>$s$ in lower case</td>
</tr>
<tr>
<td>$s$.isupper()</td>
<td>$s$ in upper case</td>
</tr>
<tr>
<td>$s$.istitle()</td>
<td>$s$ in title form</td>
</tr>
<tr>
<td>$s$.isdigit()</td>
<td>$s$ contains only digits</td>
</tr>
<tr>
<td>$s$.isalpha()</td>
<td>$s$ contains only letters</td>
</tr>
<tr>
<td>$s$.isalnum()</td>
<td>$s$ contains only letters and digits</td>
</tr>
</tbody>
</table>

Examples:

```python
x = 'hello' => x.islower() is True
  => x.isalpha() is True
  => x.isalnum() is True
```
Tests on Strings
built-in methods

For a string $s$, we have the methods

<table>
<thead>
<tr>
<th>method</th>
<th>returns True if ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$.islower()</td>
<td>$s$ in lower case</td>
</tr>
<tr>
<td>$s$.isupper()</td>
<td>$s$ in upper case</td>
</tr>
<tr>
<td>$s$.istitle()</td>
<td>$s$ in title form</td>
</tr>
<tr>
<td>$s$.isdigit()</td>
<td>$s$ contains only digits</td>
</tr>
<tr>
<td>$s$.isalpha()</td>
<td>$s$ contains only letters</td>
</tr>
<tr>
<td>$s$.isalnum()</td>
<td>$s$ contains only letters and digits</td>
</tr>
</tbody>
</table>

Examples:

$x = \text{`hello' } \Rightarrow x$.islower() is True

$\Rightarrow x$.isalpha() is True

$\Rightarrow x$.isalnum() is True
## Tests on Strings

**built-in methods**

For a string $s$, we have the methods

<table>
<thead>
<tr>
<th>method</th>
<th>returns True if ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$.islower()</td>
<td>$s$ in lower case</td>
</tr>
<tr>
<td>$s$.isupper()</td>
<td>$s$ in upper case</td>
</tr>
<tr>
<td>$s$.istitle()</td>
<td>$s$ in title form</td>
</tr>
<tr>
<td>$s$.isdigit()</td>
<td>$s$ contains only digits</td>
</tr>
<tr>
<td>$s$.isalpha()</td>
<td>$s$ contains only letters</td>
</tr>
<tr>
<td>$s$.isalnum()</td>
<td>$s$ contains only letters and digits</td>
</tr>
</tbody>
</table>

### Examples:

$x = 'hello' \Rightarrow x$.islower() is True

$\Rightarrow x$.isalpha() is True

$\Rightarrow x$.isalnum() is True
Classification of an Input String

Suppose we have a program `alphatest.py` to test if a given input is a number, is alphabetic, or is alphanumeric.

```
$ python alphatest.py
Give a number : 2341
"2341" consists of digits only

$ python alphatest.py
Give a number : hello
"hello" is alphabetic

$ python alphatest.py
Give a number : hi5
"hi5" is alphanumeric

$ python alphatest.py
Give a number : hi 5
"hi 5" fails all tests
```
Classification of an Input String

Suppose we have a program `alphatest.py` to test if a given input is a number, is alphabetic, or is alphanumeric.

```
$ python alphatest.py
Give a number : 2341
"2341" consists of digits only
```
```
$ python alphatest.py
Give a number : hello
"hello" is alphabetic
```
```
$ python alphatest.py
Give a number : hi5
"hi5" is alphanumeric
```
```
$ python alphatest.py
Give a number : hi 5
"hi 5" fails all tests
```
Classiﬁcation of an Input String

Suppose we have a program `alphatest.py` to test if a given input is a number, is alphabetic, or is alphanumeric.

```
$ python alphatest.py
Give a number : 2341
"2341" consists of digits only
```
```
$ python alphatest.py
Give a number : hello
"hello" is alphabetic
```
```
$ python alphatest.py
Give a number : hi5
"hi5" is alphanumeric
```
```
$ python alphatest.py
Give a number : hi 5
"hi 5" fails all tests
```
Classification of an Input String

Suppose we have a program `alphatest.py` to test if a given input is a number, is alphabetic, or is alphanumeric.

```
$ python alphatest.py
give a number: 2341
"2341" consists of digits only

$ python alphatest.py
give a number: hello
"hello" is alphabetic

$ python alphatest.py
give a number: hi5
"hi5" is alphanumeric

$ python alphatest.py
give a number: hi 5
"hi 5" fails all tests
```
Classification of an Input String

Suppose we have a program `alphatest.py` to test if a given input is a number, is alphabetic, or is alphanumerical.

```
$ python alphatest.py
Give a number : 2341
"2341" consists of digits only
$ python alphatest.py
Give a number : hello
"hello" is alphabetic
$ python alphatest.py
Give a number : hi5
"hi5" is alphanumerical
$ python alphatest.py
Give a number : hi 5
"hi 5" fails all tests
```
The code for `alphatest.py` is

```python
# MCS 260 Mon 15 Sep 2008 test on string

s = raw_input('Give a number: ')  # s = raw_input('Give a number: ')[1:-1]  # raw_input() is used in Python 2.7

show = '' + s + ''

if s.isalpha():
    print show + ' is alphabetic'
elif s.isdigit():
    print show + ' consist of digits only'
elif s.isalnum():
    print show + ' is alphanumeric'
else:
    print show + ' fails all tests'
```

An if elif else to test an input string

The code for `alphatest.py` is
transistors and gates
intrinsic operations

1. Digital Systems
   transistors
   logic gates

2. Intrinsic Operations
   on numbers
   on strings

3. Dictionaries and Conditionals
   writing numbers in words
   algorithm → flowchart → code

4. Summary + Assignments
Writing Numbers in Words
using dictionaries and if-elif-else statements

On a check, the amount is spelled out in words.

Program specification:
   Input: \( n \), a natural number \(< 1000\).
   Output: a string expressing \( n \) in words.

An example session with \texttt{write_numbers.py}:

```
$ python write_numbers.py
give a natural number : 125
125 is one hundred and twenty five
```
Writing Numbers in Words
using dictionaries and if-elif-else statements

On a check, the amount is spelled out in words.

Program specification:
Input: \( n \), a natural number \(< 1000\).
Output: a string expressing \( n \) in words.

An example session with write_numbers.py:

$ python write_numbers.py
give a natural number : 125
125 is one hundred and twenty five
The Dictionary
numbers spelled out in English

For all \( n \leq 20 \) and multiples of 10:

\[
d = \{ \begin{align*}
0 &: \text{`zero'}, \\
1 &: \text{`one'}, \\
2 &: \text{`two'}, \\
3 &: \text{`three'}, \\
4 &: \text{`four'}, \\
5 &: \text{`five'}, \\
6 &: \text{`six'}, \\
7 &: \text{`seven'}, \\
8 &: \text{`eight'}, \\
9 &: \text{`nine'}, \\
10 &: \text{`ten'}, \\
11 &: \text{`eleven'}, \\
12 &: \text{`twelve'}, \\
13 &: \text{`thirteen'}, \\
14 &: \text{`fourteen'}, \\
15 &: \text{`fifteen'}, \\
16 &: \text{`sixteen'}, \\
17 &: \text{`seventeen'}, \\
19 &: \text{`nineteen'}, \\
20 &: \text{`twenty'}, \\
30 &: \text{`thirty'}, \\
40 &: \text{`forty'}, \\
50 &: \text{`fifty'}, \\
60 &: \text{`sixty'}, \\
70 &: \text{`seventy'}, \\
80 &: \text{`eighty'}, \\
90 &: \text{`ninety'}, \\
100 &: \text{`hundred'}
\end{align*} \}
\]

The dictionary lookup \( d[n] \) handles special cases.
Idea for the Algorithm

case analysis

We distinguish three cases:

1. the trivial case: $n = 0$
   This is the only case we write zero.

2. large numbers $n \geq 100$
   We start writing $n/100$ hundred
   and then continue with

3. the rest: $0 < n < 100$:
   ① for $n \leq 20$: dictionary lookup
   ② for $20 < n < 100$: compute $r = n\%10$ and $n - r$
Idea for the Algorithm

case analysis

We distinguish three cases:

1. the trivial case: $n = 0$
   This is the only case we write \textit{zero}.  

2. large numbers $n \geq 100$
   We start writing $n/100$ \textit{hundred}
   and then continue with

3. the rest: $0 < n < 100$:
   1. for $n \leq 20$: dictionary lookup
   2. for $20 < n < 100$: compute $r = n\%10$ and $n - r$
Idea for the Algorithm

Case analysis

We distinguish three cases:

1. the trivial case: \( n = 0 \)
   This is the only case we write zero.

2. large numbers \( n \geq 100 \)
   We start writing \( n/100 \) hundred
   and then continue with

3. the rest: \( 0 < n < 100 \):
   
   1. for \( n \leq 20 \): dictionary lookup
   
   2. for \( 20 < n < 100 \): compute \( r = n\%10 \) and \( n - r \)
Idea for the Algorithm

case analysis

We distinguish three cases:

1. the trivial case: \( n = 0 \)
   This is the only case we write \textit{zero}.

2. large numbers \( n \geq 100 \)
   We start writing \( n/100 \) \textit{hundred}
   and then continue with

3. the rest: \( 0 < n < 100 \):
   \begin{enumerate}
   \item for \( n \leq 20 \): dictionary lookup
   \item for \( 20 < n < 100 \): compute \( r = n\%10 \) and \( n - r \)
   \end{enumerate}
transistors and gates
intrinsic operations

1. Digital Systems
   transistors
   logic gates

2. Intrinsic Operations
   on numbers
   on strings

3. Dictionaries and Conditionals
   writing numbers in words
   algorithm → flowchart → code

4. Summary + Assignments
Flowchart for `write_numbers.py`

```
n = input('Enter your number :')

n = 0?
  True  write zero

False

n > 100?
  True  write d[n/100] hundred
       n = n%100

  False

n <= 20?
  True  write d[n]

  False

r = n%10

r = 0?
  True  write d[n]
        n = 0?

  False  write d[n-r] + d[r]
```
Flowchart for write_numbers.py

\[
\text{n = input('Enter your number :')}\]

- \(n = 0?\) True → write zero
- \(n = 0?\) False → \(n > 100?\) True → write \(d[\frac{n}{100}]\) hundred
  \(n = n\%100\)
  - \(n \leq 20?\) True → write \(d[n]\)
  - \(r = n\%10\)
  - \(r = 0?\) True → write \(d[n]\)
  - False → write \(d[n-r] + d[r]\)
- \(n > 100?\) False → \(n = n\%100\)
- \(n = 0?\) True → write zero
- \(n = 0?\) False → False
Flowchart for `write_numbers.py`

```python
n = input('Enter your number :')

if n == 0:
    write zero
else:
    if n > 100:
        write \[\frac{n}{100}\] hundred
        n = n%100
    else:
        if n <= 20:
            write \[\text{d}[n]\]
        else:
            r = n%10
            if r == 0:
                write \[\text{d}[n]\]
            else:
                write \[\text{d}[n-r] + \text{d}[r]\]
```

Flowchart for write_numbers.py

```
n = input('Enter your number :')

if n == 0:
    write zero

if n > 100:
    write d[\frac{n}{100}] hundred
    n = n%100

if n <= 20:
    write d[n]

r = n%10

if r == 0:
    write d[n]
else:
    write d[n-r] + d[r]
```
Flowchart for write_numbers.py

```
n = input('Enter your number :

if n == 0:
    write zero

if n > 100:
    write d[n/100] hundred
    n = n%100

if n <= 20:
    write d[n]

r = n%10

if r == 0:
    write d[n]

write d[n-r] + d[r]
```
First Half of write_numbers.py

The code starts with the dictionary \( d = \ldots \)

\[
n = \text{input}('\text{give a natural number : }')
\]
\[
\text{outcome} = '%d is ' \% n
\]
\[
\text{if } n == 0: \quad \quad \quad \# \text{trivial case}
\]
\[
\text{outcome} += d[n]
\]
\[
\text{elif } n >= 100:
\quad \text{outcome} += d[n/100] \quad + \quad ' and ' + d[100]
\]
\[
n = n \% 100
\]
\[
\text{if } n != 0:
\quad \text{outcome} += ' and '
\]

This handles the first two cases of the algorithm.
First Half of `write_numbers.py`

The code starts with the dictionary `d = ...`

```python
n = input('give a natural number : ')  
outcome = '%d is ' % n  
if n == 0:  # trivial case  
    outcome += d[n]  
elif n >= 100:  
    outcome += d[n/100] + ' ' + d[100]  
    n = n % 100  
if n != 0:  
    outcome += ' and '  
```

This handles the first two cases of the algorithm.
Second Half of `write_numbers.py`

We continue with the rest $0 \leq n < 100$:

```python
if n > 0:        # write zero only once
    if n <= 20:
        outcome += d[n]
    else:
        r = n % 10
        if r == 0:
            outcome += d[n]
        else:
            outcome += d[n-r] + ' ' + d[r]
print outcome
```
Second Half of `write_numbers.py`

We continue with the rest $0 \leq n < 100$:

```python
if n > 0:  # write zero only once
    if n <= 20:
        outcome += d[n]
    else:
        r = n % 10
        if r == 0:
            outcome += d[n]
        else:
            outcome += d[n-r] + ' ' + d[r]
print outcome
```
Summary + Assignments

In this lecture we covered more of

- section 1.1 in *Computer Science. An Overview*
- chapter 5 in *Python Power!*

Assignments:

1. Draw all transistors needed to realize an OR gate and describe its working.
2. Construct truth tables for
   1. (A OR B) OR NOT (A AND B)
   2. NOT ((A OR C) OR B) OR (A AND C)
3. Draw the logic gates to realize the expressions of the previous exercise.
4. Let secret be a secret number the user of a Python program has to guess. Give code for prompting the user for a guess and for printing feedback.