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
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.
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

electronic circuits to represent bits

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 $\rightarrow$ flowchart $\rightarrow$ code

4. Summary + Assignments
Logic Gates

Logic gates are circuits that correspond to logic operators.

Representations of NOT, AND, OR:

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

\[
x \text{ NOR } y \quad = \quad \text{NOT} \ (x \ \text{OR} \ y)
\]
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 NAND Gate

Input voltages $V_1$ and $V_2$

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

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

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

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

Intro to Computer Science (MCS 260)
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.
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
### Conversions for Numbers (Built-in Functions)

<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>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)
```
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'

>>> '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.
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:

\[
x = 'hello' \implies x.islower() \text{ is True}
\]

\[
\implies x.isalpha() \text{ is True}
\]

\[
\implies x.isalnum() \text{ is True}
\]
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.

```bash
$ 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
```
an if elif else to test an input string

The code for `alphatest.py` is

```python
DATA = input('Give a number : ')  
SHOW = '"' + DATA + '"'  
if DATA.isalpha():  
    print(SHOW + ' is alphabetic ')  
elif DATA.isdigit():  
    print(SHOW + ' consist of digits only')  
elif DATA.isalnum():  
    print(SHOW + ' is alphanumeric ')  
else:  
    print(SHOW + ' fails all tests')
```
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
writing numbers in words – applying dictionaries

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
the dictionary: numbers spelled out in English

For all $n \leq 20$ and multiples of 10:

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

The dictionary lookup $\text{DIC}[n]$ 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 \( n - r \)
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]

False
write d[n-r] + d[r]
```
The code starts with the dictionary `DIC = ...`

DATA = input('give a natural number : ')
NBR = int(DATA)
OUTCOME = '%d is ' % NBR
if NBR == 0:
    OUTCOME += DIC[NBR]
elif NBR >= 100:
    OUTCOME += DIC[NBR/100] + ' ' + DIC[100]
    NBR = NBR % 100
if NBR != 0:
    OUTCOME += ' and '

This handles the first two cases of the algorithm.
We continue with the rest $0 \leq n < 100$:

```python
if NBR > 0:  # write zero only once
    if NBR <= 20:
        OUTCOME += DIC[NBR]
    else:
        REST = NBR % 10
        if REST == 0:
            OUTCOME += DIC[NBR]
        else:
            OUTCOME += DIC[NBR-REST] + ' ' + DIC[REST]

print(OUTCOME)
```
Assignments

1. Draw all transistors needed to realize an OR gate and describe its working.

2. Construct truth tables for
   1. \((A \text{ OR } B) \text{ OR } \neg (A \text{ AND } B)\)
   2. \(\neg ((A \text{ OR } C) \text{ OR } B) \text{ OR } (A \text{ AND } C)\)

3. Draw the logic gates to realize the expressions of the previous exercise.

4. Let \texttt{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.

5. Write a script to use \texttt{dbm} to store the dictionary \(d\) to spell numbers out in English.

6. Modify the \texttt{write_numbers.py} program so it uses the \texttt{dbm} file made in the previous exercise.
In this lecture we covered more of

- section 1.1 in *Computer Science. An Overview*
- pages 135-138 of *Python Programming*