Outline

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

MCS 260 Lecture 9
Introduction to Computer Science
Jan Verschelde, 1 February 2016

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

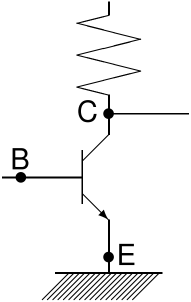
1. Digital Systems
   - transistors
   - logic gates
2. Intrinsic Operations
   - on numbers
   - on strings
3. Dictionaries and Conditionals
   - writing numbers in words
   - algorithm → flowchart → code
4. Summary + Assignments

Transistors

electronic circuits to represent bits

Transistors have three connections to the outside:

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



High Voltage: 1

Low Voltage: 0

transistors and gates
intrinsic operations

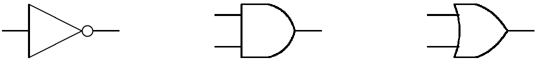
1. Digital Systems
   - transistors
   - logic gates
2. Intrinsic Operations
   - on numbers
   - on strings
3. Dictionaries and Conditionals
   - writing numbers in words
   - algorithm $\rightarrow$ flowchart $\rightarrow$ code
4. Summary + Assignments

Logic Gates

implement logic operators

Logic gates are circuits that correspond to logic operators.

Representations of NOT, AND, OR:

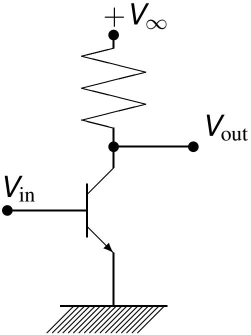


x NAND y
= NOT (x AND y)

x NOR y
= NOT (x OR y)

A NOT Gate

as realized by a transistor



Input Voltage V_{in}

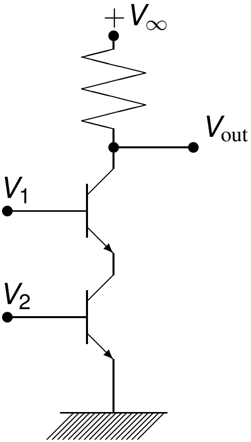
$V_{\text{in}} = $ low
$\Rightarrow$ switch is open
$\Rightarrow$ $V_{\text{out}} = +V_\infty$

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

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

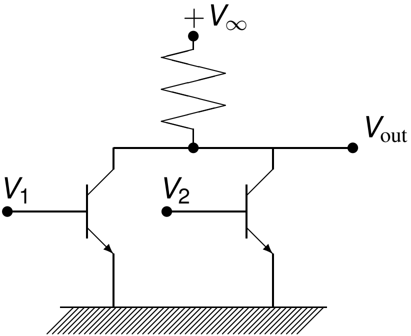
If both V_1 and V_2 are high:

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

intrinsic operations

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

For every variable x, the function

id(x) returns the address of x,

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)

function	converts ...
`int()`	string or number to integer
`float()`	string or number to float
`complex()`	string or number to complex number

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

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

transistors and gates
intrinsic operations

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.

tests on strings

built-in methods

For a string `s`, we have the methods

method	returns True if ...
`s.islower()`	`s` in lower case
`s.isupper()`	`s` in upper case
`s.istitle()`	`s` in title form
`s.isdigit()`	`s` contains only digits
`s.isalpha()`	`s` contains only letters
`s.isalnum()`	`s` contains only letters and digits

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
```

an if elif else to test an input string

The code for alphatest.py is

```
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 `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:

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

The dictionary lookup `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`.

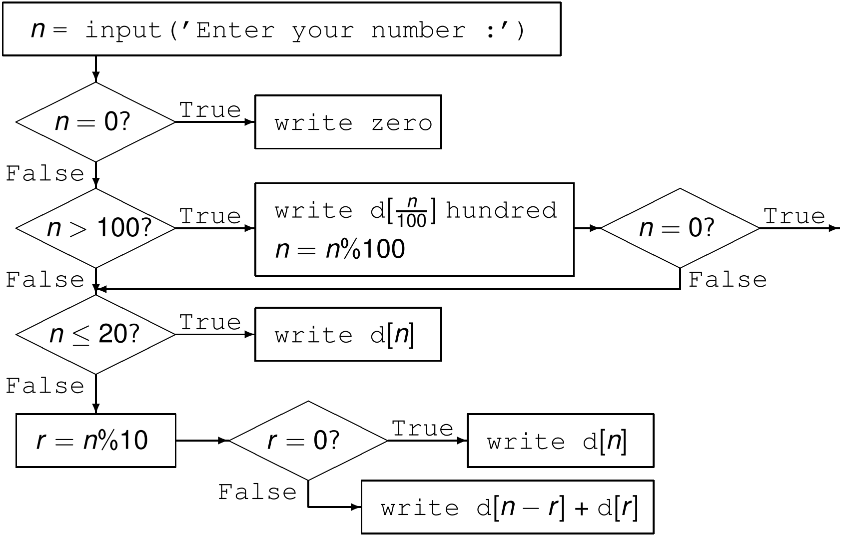
2. large numbers $n \geq 100$
   We start writing $n/100$ `hundred`
   and then continue with

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

flowchart for `write_numbers.py`



first half of `write_numbers.py`

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.

second half of write_numbers.py

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

```
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 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.
5. Write a script to use `dbm` to store the dictionary `d` to spell numbers out in English.
6. Modify the `write_numbers.py` program so it uses the `dbm` file made in the previous exercise.

Reading Materials

In this lecture we covered more of

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