Welcome to MCS 275

MCS 275 Lecture 1
Programming Tools and File Management
Jan Verschelde, 9 January 2017

Welcome to MCS 275

Content of the Course

recommended Text Books

(1) J.G. Brookshear *Computer Science. An Overview*. 9th or 10th Edition Addison-Wesley 2007, 2009.
→ intro to CS, examples of C and C++

(2) B.N. Miller and D.L. Ranum: *Python Programming in Context*. Jones and Bartlett 2009.
→ good introduction to Python for mcs 260,
for mcs 275: chapters 7 to 9

The material between midterms is still based on
Rashi Gupta: *Making Use of Python*. Wiley 2002.
Overview:

1. Python glues software components
2. programming techniques, e.g.: recursion
3. applications to science & engineering
4. algorithms and data structures

Revised Catalog Description for mcs 275

still in progress...

Programming in Python, graphical user interfaces.

Recursive problem solving, enumeration and backtracking.

Recursive data structures, divide and conquer.

Programming with CGI, MySQL, sockets, threads,
and web servers.

Welcome to MCS 275

Prerequisites & Expectations

MCS 275 follows MCS 260

What you should know already:

1. basic computer/computing literacy; and
2. elements of algorithm and program design.

Programming involves understanding of specifications, design of solution methods, definition of algorithms,
and coding (or implementation).

Only coding depends on a programming language.

Programming is a skill, we **learn by doing**.
Therefore,

1. emphasis on five computer projects
2. active participation in the lab sessions

Welcome to MCS 275

1. About the Course
   - Course Content
   - Prerequisites & Expectations
2. Introduction to Programming
   - Scripting in Python
   - from OOP to LAMP
   - example: Factorization in Primes
3. Summary

Scripting in Python

programming in the small

- rapid prototyping
- use at command prompt (the “shell”)
  or in integrated development environment (IDLE)
- Python is interpreted
  - \+ runs immediately
  - \+ dynamic typing and garbage collection
  - − not as efficient as compiled code
- Python is open to C extensions
  → computationally intensive portions in C

We will use Python 3.6 in this course.

Welcome to MCS 275

from OOP to LAMP

programming in the large

- OOP = Object Oriented Programming
  $\rightarrow$ organization of code to promote
  reuse, change, and scalability
- The Open Source revolution has given us

  L Linux, operating system
  A Apache, web server
  M MySQL, database
  P Python, scripting language

  Python glues software components

Welcome to MCS 275

1. About the Course
   - Course Content
   - Prerequisites & Expectations
2. Introduction to Programming
   - Scripting in Python
   - from OOP to LAMP
   - example: Factorization in Primes
3. Summary

Factorization in Primes

an example program

A prime has only two divisors: 1 and itself.

Every natural number can be factored
as a unique product of primes.

Problem (input/output specification):

input	:	a natural number n
output	:	a list of primes $[p_1, p_2, \ldots, p_k]$, $n = p_1 \times p_2 \times \cdots \times p_k$

A related (easier) problem:

input	:	a natural number n
output	:	a list of all divisors of n

Let us look at the easier problem first.

Enumerating all Divisors

Problem (input/output specification):

input : a natural number n

output : a list of all nontrivial divisors of n
(1 and n are trivial divisors)

first, we care about clarity and correctness

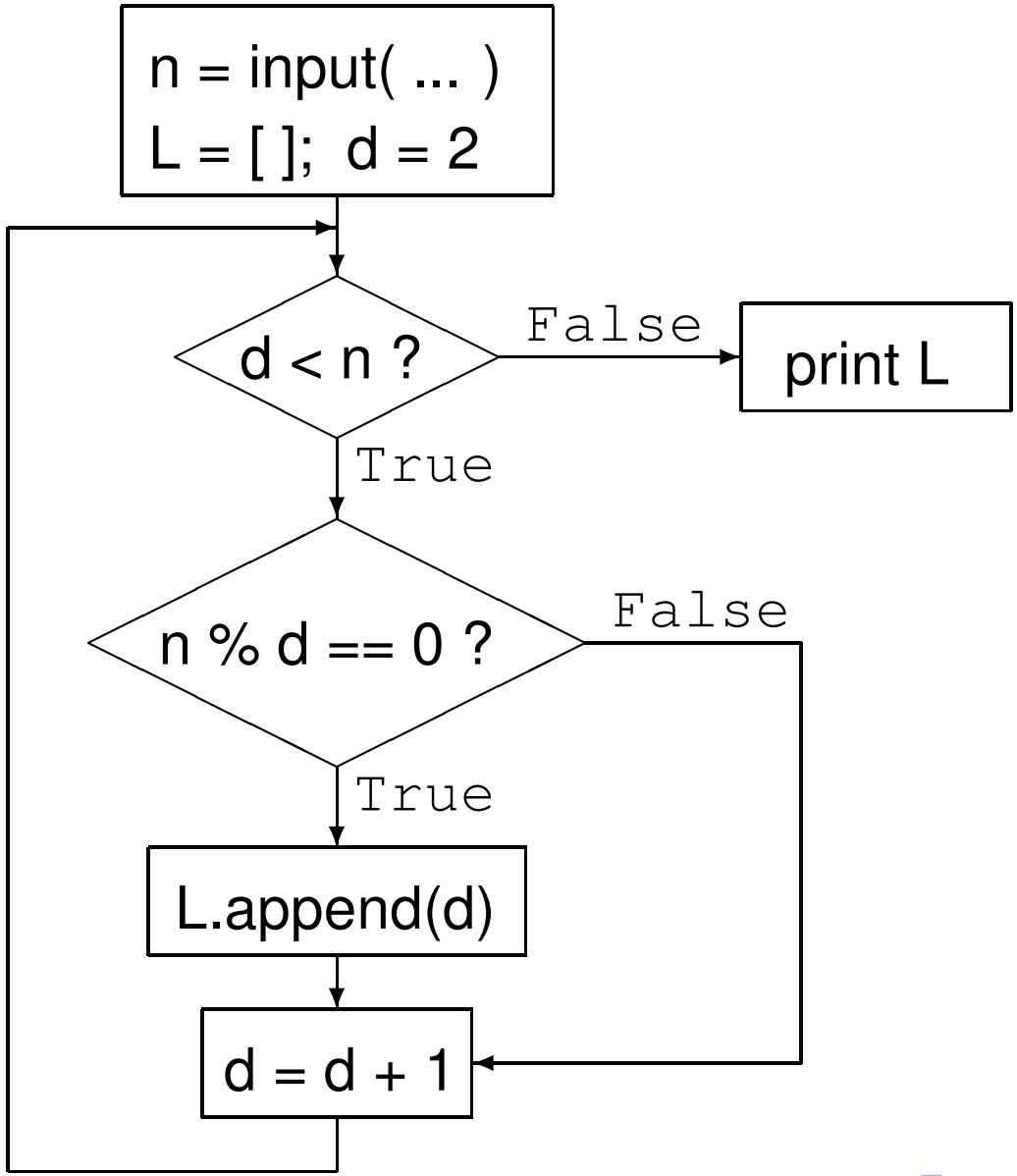
Ingredients in a *first* algorithm:

1. d divides n if $n \,\%\, d == 0$ (zero remainder)
2. enumerate all candidate divisors,
   for d ranging between 2 and $n - 1$, `for d in range(2, n)`
3. append divisor d to a list `L`: `L.append(`d`)`

Now we will put the ingredients into a recipe.

Flowchart

enumerating all divisors



Python code in the file `enumdivs.py`

```
# MCS 275 L-1 Mon 9 Jan 2017 : enumdivs.py
"""
enumerate all nontrivial divisors of a number
"""

NBR = int(input('give a number : '))
FACTORS = []
for DIVISOR in range(2, NBR):
    if NBR % DIVISOR == 0:
        FACTORS.append(DIVISOR)
print('nontrivial divisors of %d :' % NBR, FACTORS)
```

Running the script at the command prompt `$`

```
$ python enumdivs.py
give a number : 24
nontrivial divisors of 24 : [2, 3, 4, 6, 8, 12]
```

Some Observations

on `enumdivs.py`

- the `input()` command returns a string
  with `int()` we cast the string into an integer.
- `range(2, NBR)` is `[2,3,..,NBR-1]`
- all code in the same `for` or `if` must be preceded by same number of spaces
- three different occurrences of %:
  1. `NBR % DIVISOR`: remainder after division by `DIVISOR`
  2. `%d`: formatting code for decimal output
  3. `%`: format conversion operator in `print`
- `append` is a method for the list data type
- *long* numbers will take a *long* time

Factorization into Primes

back to the original problem

Problem (input/output specification):

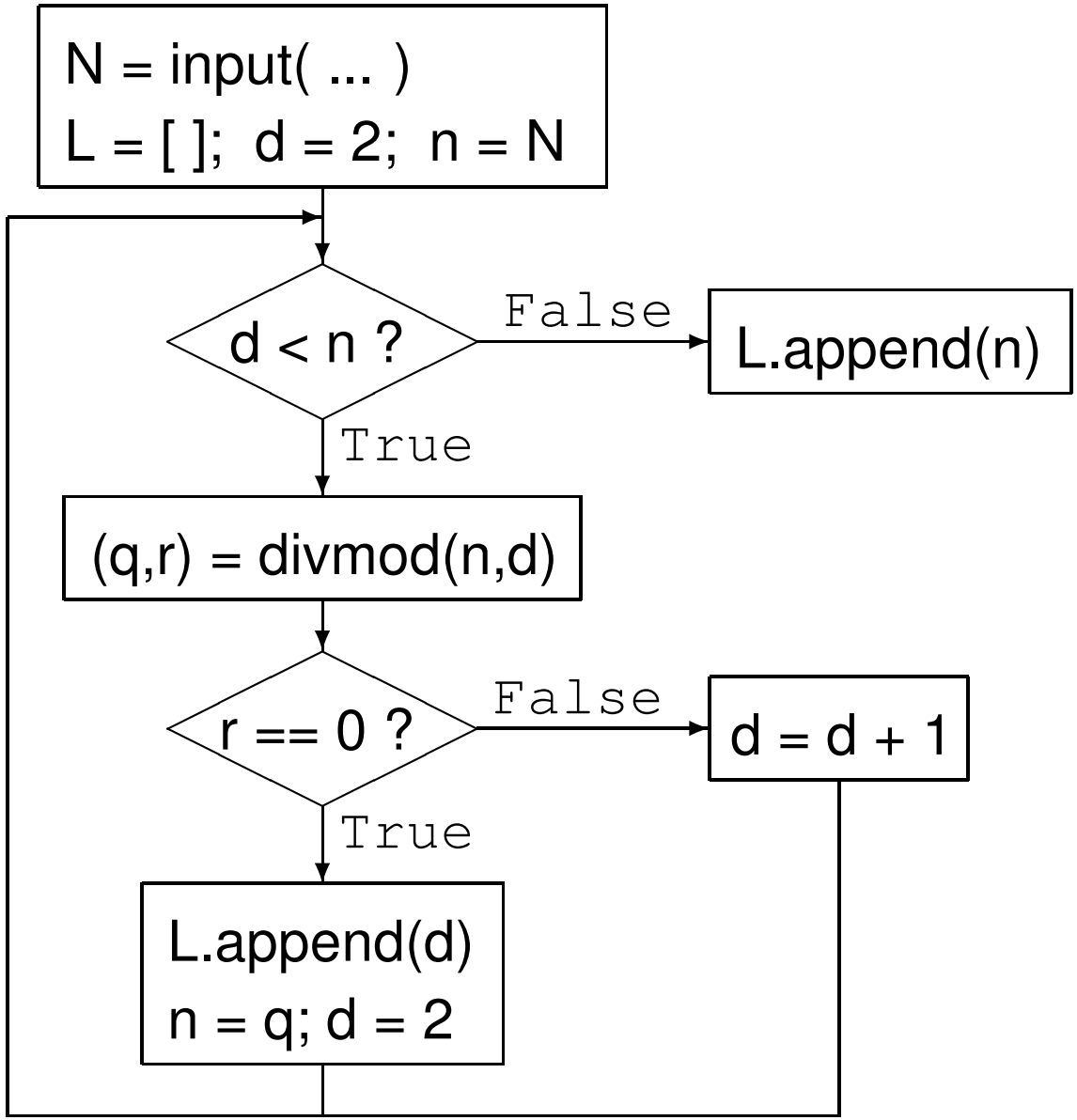
input : a natural number n

output : a list of primes $[p_1, p_2, \ldots, p_k]$,
$n = p_1 \times p_2 \times \cdots \times p_k$

Ingredients in a *first* algorithm:

1. we enumerate all candidate divisors, starting at 2
2. when we find a divisor d of n, we continue with the quotient q, so we assign $n = q$
3. $(q, r) = \texttt{divmod}(n, d)$ computes quotient q and remainder r of the division of n by d
4. all divisors are appended to a list

Flowchart for Factoring in Primes



the script `facnum.py`

```
from functools import reduce
NBR = int(input('give a natural number n : '))
FACTORS = []
DIVISOR = 2
WORK = NBR
while DIVISOR < WORK:
    (QUOT, REST) = divmod(WORK, DIVISOR)
    if REST == 0:
        FACTORS.append(DIVISOR)
        (WORK, DIVISOR) = (QUOT, 2)
    else:
        DIVISOR = DIVISOR + 1
FACTORS.append(WORK)
print('factors of ' + str(NBR) + ' :', FACTORS)
PRD = reduce(lambda x, y: x*y, FACTORS)
print('product of factors :', PRD)
```

Functional Programming

The statement

```
PRD = reduce(lambda x,y: x*y , FACTORS)
```

computes the product

$$p = \prod_{p_i \in L} p_i = p_1 \times p_2 \times \cdots \times p_k.$$

The anonymous function `lambda x,y: x*y`
is applied to the list `FACTORS` reducing the list `FACTORS`
to the product of all its elements.

Exercises

1. Use `facnum.py` to verify that 2017 is a prime number. What is the next year (following 2017) that is a prime number?
2. Make `enumdivs.py` more efficient by examining only $\sqrt{n}$ candidate divisors.
3. Relate the number of $n\%d$ operations in `enumdivs.py` to the number of decimal places of n.
   If we add one more decimal place to the input n,
   how many more operations does `enumdivs.py` do?
4. Improve the efficiency of `facnum.py` also by taking fewer candidate divisors as in exercise 2 above.
5. Verify the correctness of `facnum.py` by an exhaustive enumeration of all numbers n within the range $1, \ldots, m$, where m is given by the user.
6. Analyze the running time of `facnum.py` experimentally by trying various input numbers. For which numbers does `facnum.py` take the longest time? Which numbers are easiest?

Summary

In this lecture we covered

1. write algorithms from i/o specifications;
2. encode algorithms in Python scripts.

Lab sessions take place in SEL 2263 (Mac Lab).

Topics of the lab sessions this week:

1. run scripts `enumdivs.py` and `facnum.py`
2. consider exercises of the lectures
3. small quiz in the last half hour