Outline

MCS 260 Lecture 5
Introduction to Computer Science
Jan Verschelde, 22 January 2016

Operating Systems
Turtle Graphics

The Operating System (OS)

needed to operate a computer

- software layer between users and physical machine
- single user for Personal Computer (PC) or
  multi user for workstation or mainframe
- realization of a **virtual machine**: give user illusion that computer is fully dedicated to the user
- MS-DOS and UNIX are commonly used
  via MS Windows or X Windows respectively
  (although MS Windows has replaced MS-DOS)
- goals: satisfy users and maximize performance

Layers in an OS

Operating System functions

user programs

5. command interpreter (shell)
   activate user programs
4. file system
   control file access

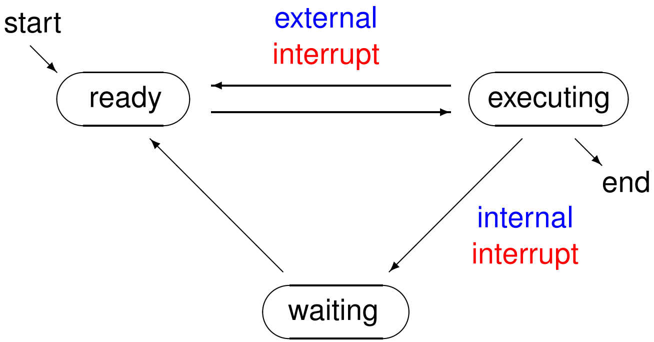
$$\left.\begin{array}{l}\text{3. peripherals management} \\ \quad\text{manage i/o devices} \\ \text{2. memory management} \\ \quad\text{allocate and free memory} \\ \text{1. process management} \\ \quad\text{control program execution}\end{array}\right\}\begin{array}{c}\text{kernel} \\ \text{of OS}\end{array}$$

physical machine

Process Management

a process evolves in time

Three states of a process:



- external interrupt: end of time slice
- internal interrupt: program does input or output

Memory Management

allocate memory for each process

pagination	is the mechanism to partition the main memory into pages, each page is a contiguous memory area, of fixed size
segmentation	is the division of a program, separating parts that perform different functions
virtual memory	is the term used to describe the impression the memory manager gives
swap space	is that part of the disc used to swap memory pages onto disc for memory intensive jobs

Peripherals and File Management

Drivers provide mechanisms to communicate data to and from the peripherals. We distinguish

- physical drivers are part of the equipment;
- logical drivers belong to the OS.

The file system is the organization of mass storage:

- Files are collected in directories.
- Directories are organized in a tree structure.

Operating Systems
Turtle Graphics

1. Operating Systems
   - overview & concepts
   - files and processes in UNIX
   - the platform and os modules

2. Turtle Graphics
   - importing the turtle module
   - drawing a spiral
   - colors with turtle

3. Summary + Assignments

The UNIX Operating System

a little history

- Originally written in C by Ken Thompson and Dennis Ritchie at AT&T in the early seventies.
- Further development of `vi` and `C shell`
  into BSD UNIX, BSD = Berkeley Software Distribution.
- Many variants of UNIX by IBM, DEC, HP, Sun.
- Standards provided by POSIX: Portable Operation System Interface for Computer Environments.
- Linux is distributed under the GNU General Public License: mandatory to make the source code public.

Linus Torvalds

original developer of the Linux kernel

+ original developer of the Linux (the x refers to Unix) kernel, decides which code goes into it.
+ Linus's Law: *Given enough eyeballs, all bugs are shallow.* Linux is Free and Open Source.
+ Git originated from Linux kernel developers, including Linus Torvalds. Git is used to manage source code.

image taken from `http://github.com/torvalds`.

processes in UNIX

UNIX is a multi-user system:

- system administration is done by `root`
- users have restricted privileges

Some useful commands to control processes:

w	lists current users, displays load
ps	list processes
kill	halt or stop execution

Do `man` followed by a command name to see the manual pages for that command.

the file system in UNIX

After logging in, by default, the current directory is your home directory where your files are stored.

Some useful commands on files

cp	copy a file, duplicates
mv	move a file, renames
rm	remove a file, deletes

Some commands on directories:

pwd	print working directory
ls	list content of current directory
cd	change directory, `cd ..` goes up in the tree
mkdir	make a directory
rmdir	remove a directory

UNIX in MCS 260

how we experience UNIX

1. Computers in SEL 2263 run Mac OS X.
   Mac OS X is based on the BSD UNIX implementation.
   Via **Terminal** we get a shell as on any UNIX.
2. Cygwin is a Linux-like environment for Windows
   It *emulates* (behaves like) the UNIX OS.
   You can install Cygwin on your Windows PC at home,
   or install Linux and enable dual boot.
3. Knoppix is based on Debian GNU/Linux, *on CD*.
   You can run Linux on most PCs without installation.

Operating Systems
Turtle Graphics

The platform Module – get information about your OS

Using the `platform` module,
we obtain platform-dependent data in a Python session:

```
>>> import platform
>>> platform.machine()
'x86_64'
>>> platform.architecture()
('64bit', '')
>>> platform.processor()
'i386'
```

More Information about your OS

the session continued ...

```
>>> platform.python_compiler()
'GCC 4.2.1 (Apple Inc. build 5666) (dot 3)'
>>> platform.system()
'Darwin'
>>> platform.uname()
('Darwin', 'asterix.local', '13.1.0',
 'Darwin Kernel Version 13.1.0: Wed Apr  2
  23:52:02 PDT 2014;
 root:xnu-2422.92.1~2/RELEASE_X86_64',
 'x86_64', 'i386')
```

Application: platform-***in***dependent code.

the os module: operating system independence

The module `os` is an interface to the operating system.
To execute any command, use `os.system`:

```
>>> import os
>>> os.system('ls')
```

The module `os` exports functions on files and directories, independent of specific operating system commands.

To list the content of a directory:

```
os.listdir( < directory name > )
```

on return is a list of file names (as strings).
For example, for the current directory:

```
>>> os.listdir('.')
```

Runs on Unix, MacOS X, and Windows: portable.

Portability in Python via the module `os`

We say that a program is *portable* if it runs unmodified on different operating systems and architectures.

In a Python session we may access OS functions via the methods of the `os` module, some examples:

```
>>> import os
>>> os.uname()         # identifies current OS
>>> os.listdir('.')    # same as ls on UNIX
>>> os.getcwd()        # get current directory
>>> os.chdir('/tmp')   # change directory to /tmp
>>> help(os)           # for more information
```

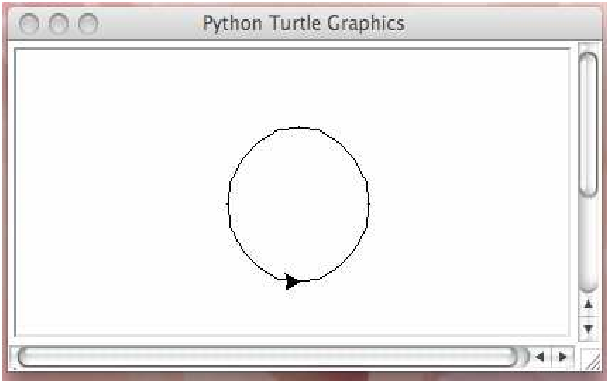
Using the methods of the `os` module we can build portable system utilities, e.g.: find a file on a disk.

Operating Systems
Turtle Graphics

Turtle Graphics: `turtle` is a builtin module

For example, to draw a circle of radius 50 pixels:

```
>>> from turtle import Turtle
>>> sam = Turtle()
>>> sam.circle(50)
```



modules, classes, and objects

Explaining the three statements on the previous slide:

1. We import the class `Turtle` from the module `turtle`:

   ```
   >>> from turtle import Turtle
   ```

   A module is a library of code. A class defines a data type.

2. We then make an object `sam` of the class `Turtle`:

   ```
   >>> sam = Turtle()
   ```

   After the assigment, `sam` refers to a `Turtle` object.

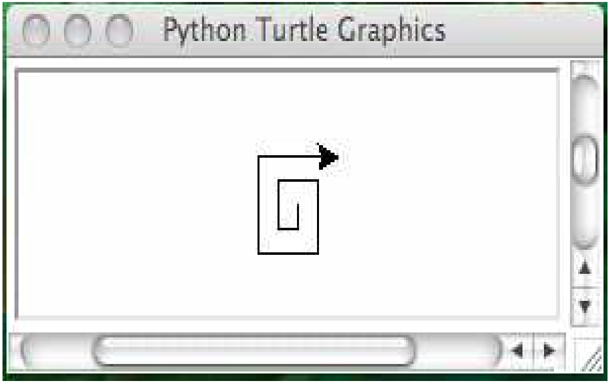
3. Calling a method of the class `Turtle`:

   ```
   >>> sam.circle(50)
   ```

   The `circle` method is defined for any object of the `Turtle` class.

Operating Systems
Turtle Graphics

the start of a spiral



some useful commands

command	description
`from turtle import Turtle`	imports `Turtle` from the `turtle` module
`myTurtle = Turtle()`	make an object `myTurtle`

`myTurtle.up()`
`myTurtle.down()`
	put the pen up
	put the pen down

`myTurtle.left(angle)`
`myTurtle.right(angle)`
	turn left by angle given in degrees
	turn right by angle in degrees

`myTurtle.forward(d)`	go forward by `d` pixels

drawing a spiral with a script

```
from turtle import Turtle
LIZ = Turtle()
LIZ.down()        # put the pen down
LIZ.right(90)     # turn 90 degrees to the right
LIZ.forward(10) # go forward by 10 pixels
LIZ.right(90)     # turn 90 degrees to the right
LIZ.forward(10) # go forward by 10 pixels
LIZ.right(90)     # turn 90 degrees to the right
LIZ.forward(20) # go forward by 20 pixels
LIZ.right(90)     # turn 90 degrees to the right
LIZ.forward(20) # go forward by 20 pixels
LIZ.right(90)     # turn 90 degrees to the right
LIZ.forward(30) # go forward by 30 pixels
LIZ.right(90)     # turn 90 degrees to the right
LIZ.forward(30) # go forward by 30 pixels
LIZ.right(90)     # turn 90 degrees to the right
LIZ.forward(40) # go forward by 40 pixels
LIZ.right(90)     # turn 90 degrees to the right
LIZ.forward(40) # go forward by 40 pixels
ANS = input('hit enter to exit') # leave window
```

more turtle commands

Functions exported by the `turtle` module:

command	description
`window_width()`	#pixels in width of turtle window
`window_height()`	#pixels in height of turtle window
`tracer(False)`	do not animate arrow to draw faster by default: `turtle.tracer(True)`

Methods applicable to any `Turtle` object:

command	description
`position()`	returns the position of the turtle
`goto(x, y)`	goes to position `(x, y)` and draws a line from the current position to `(x, y)` if pen is down

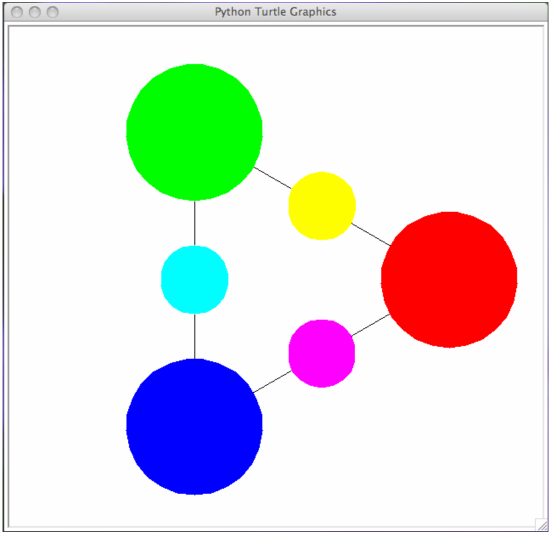
a script to draw the spiral

```
from turtle import Turtle        # import the Turtle class
SAM = Turtle()                   # SAM is the name of a Turtle object
SAM.up()                         # put the pen up
SAM.goto(0, 0)                   # goto center
SAM.down()                       # put the pen down
POS = SAM.pos()                  # get the current position
SAM.goto(POS[0], POS[1] - 10)    # go 10 pixels down
POS = SAM.pos()
SAM.goto(POS[0] - 10, POS[1])    # go 10 pixels to the left
POS = SAM.pos()
SAM.goto(POS[0], POS[1] + 20)    # go 20 pixels up
POS = SAM.pos()
SAM.goto(POS[0] + 20, POS[1])    # go 20 pixels to the right
POS = SAM.pos()
SAM.goto(POS[0], POS[1] - 30)    # go 30 pixels down
POS = SAM.pos()
SAM.goto(POS[0] - 30, POS[1])    # go 30 pixels to the left
POS = SAM.pos()
SAM.goto(POS[0], POS[1] + 40)    # go 40 pixels up
POS = SAM.pos()
SAM.goto(POS[0] + 40, POS[1])    # go 40 pixels to the right
ANS = input('hit enter to exit') # else window disappears
```

Operating Systems
Turtle Graphics

1. Operating Systems
   - overview & concepts
   - files and processes in UNIX
   - the platform and os modules
2. Turtle Graphics
   - importing the turtle module
   - drawing a spiral
   - colors with turtle
3. Summary + Assignments

colors with turtle



rgb code for colors

Humans are trichromats, we have 3 types of color receptors.

The rgb code defines colors via triplets:

r	(1.0, 0.0, 0.0)	pure red
g	(0.0, 1.0, 0.0)	pure green
b	(0.0, 0.0, 1.0)	pure blue

(0.0, 0.0, 0.0) is black, (1.0, 1.0, 1.0) is white,
and (x, x, x), for $0 < x < 1$, is some shade of grey.

Taking numbers less than 1.0 changes the intensity,
e.g.: (0.6, 0.0, 0.0) gives a different shade of red.

Combining the red, green, and blue intensities allows for all other colors, e.g. (1.0, 1.0, 0.0) gives yellow.

setting colors with turtle

Let `myTurtle` be an object of the class `Turtle`.

We define the color of the pen with `penpencolor`, e.g.:

```
myTurtle.pencolor((1.0, 0.0, 0.0))
```

makes that lines drawn will show up in red.

The color used to fill enclosed drawings is set as

```
myTurtle.fillcolor((0.0, 1.0, 0.0))
```

We can set both `fillcolor` and `pencolor` at once, e.g.:

```
myTurtle.color((0.0, 1.0, 0.0), (0.0, 1.0, 0.0))
```

drawing a triangle

```
from math import cos, sin, pi

from turtle import Turtle

ANGLE = 2*pi/3 # angle to draw a triangle
BIGRAD = 200   # radius of the enscribed circle
CIRRAD = 80    # radius of the colored circles

# first we draw the triangle

TRIANGLE = Turtle()
TRIANGLE.up()
TRIANGLE.goto(BIGRAD, 0)
TRIANGLE.down()
TRIANGLE.goto(BIGRAD*cos(ANGLE), BIGRAD*sin(ANGLE))
TRIANGLE.goto(BIGRAD*cos(2*ANGLE), BIGRAD*sin(2*ANGLE))
TRIANGLE.goto(BIGRAD, 0)
ANS = input("hit enter to continue ...")
```

red, green, and blue disks

```
# at the corners we put primary rgb colored disks
TRIANGLE.tracer(False)
TRIANGLE.up()
TRIANGLE.goto(BIGRAD, -CIRRAD)
TRIANGLE.down()
TRIANGLE.begin_fill()
TRIANGLE.color((1.0, 0.0, 0.0), (1.0, 0.0, 0.0))
TRIANGLE.circle(CIRRAD)
TRIANGLE.end_fill()
TRIANGLE.up()
TRIANGLE.goto(BIGRAD*cos(ANGLE), BIGRAD*sin(ANGLE)-CIRRAD)
```

Notice the `-CIRRAD` in the second coordinate...

script continues ...

```
TRIANGLE.down()
TRIANGLE.begin_fill()
TRIANGLE.color((0.0, 1.0, 0.0), (0.0, 1.0, 0.0))
TRIANGLE.circle(CIRRAD)
TRIANGLE.end_fill()
TRIANGLE.up()
TRIANGLE.goto(BIGRAD*cos(2*ANGLE), \
    BIGRAD*sin(2*ANGLE)-CIRRAD)
TRIANGLE.down()
TRIANGLE.begin_fill()
TRIANGLE.color((0.0, 0.0, 1.0), (0.0, 0.0, 1.0))
TRIANGLE.circle(CIRRAD)
TRIANGLE.end_fill()
ANS = input("hit enter to continue ...")
```

combined colors

```
# then we draw the combinations at the edges
ANGLE2 = pi/3 # smaller angle for 6-gon
EDGE = 100    # to put circles on edges of triangle
RAD2 = 40     # radius of the secondary rgb colored disks
TRIANGLE.up()
TRIANGLE.goto(EDGE*cos(ANGLE2), EDGE*sin(ANGLE2)-RAD2)
TRIANGLE.down()
TRIANGLE.begin_fill()
TRIANGLE.color((1.0, 1.0, 0.0), (1.0, 1.0, 0.0))
TRIANGLE.circle(RAD2)
TRIANGLE.end_fill()
```

script continues ...

```
TRIANGLE.up()
TRIANGLE.goto(EDGE*cos(ANGLE+ANGLE2), \
    EDGE*sin(ANGLE+ANGLE2)-RAD2)
TRIANGLE.down()
TRIANGLE.begin_fill()
TRIANGLE.color((0.0, 1.0, 1.0), (0.0, 1.0, 1.0))
TRIANGLE.circle(RAD2)
TRIANGLE.end_fill()
TRIANGLE.up()
TRIANGLE.goto(EDGE*cos(2*ANGLE+ANGLE2), \
    EDGE*sin(2*ANGLE+ANGLE2)-RAD2)
TRIANGLE.down()
TRIANGLE.begin_fill()
TRIANGLE.color((1.0, 0.0, 1.0), (1.0, 0.0, 1.0))
TRIANGLE.circle(RAD2)
TRIANGLE.end_fill()
# to prevent the turtle window from disappearing
ANS = input('hit enter to exit ...')
```

Summary + Assignments

In this lecture we covered

- sections 3.1,2,3 of *Computer Science. An Overview*
- §1.5.3 of *Python Programming in Context*

Assignments:

1. Find out what `Cygwin` and `Knoppix` are.
2. Consider installing Linux on your PC.
3. Use turtle to draw a regular pentagon and fill it with your favorite color.
4. Draw the big letters T and E using turtle.
5. Make a smiley face with turtle.