INFT1004 - SEMESTER 1 - 2017		LECTURE TOPICS	
Week 1	Feb 27	Introduction, Assignment, Arithmetic	
Week 2	Mar 6	Sequence, Quick Start, Programming Style	
Week 3	Mar 13	Pictures, Functions, Media Paths	
Week 4	Mar 20	Arrays, Pixels, For Loop, Reference Passing	
Week 5	Mar 27	Nested Loops, Selection, Advanced Pictures	
Week 6	Apr 3	Lists, Strings, Input & Output, Files	Practical Test
Week 7	Apr 10	Drawing Pictures, Program Design, While Loop	Assignment set
Recess	**Apr 14 – Apr 23**	**Mid Semester Recess Break**	
Week 8	Apr 24	No Lecture / Revision and Assignment in Labs	
Week 9	May 1	Data Structures, Processing sound	
Week 10	May 8	Advanced sound	Assignment part 1 due 8:00am Tue, May 9
Week 11	May 15	Movies, Scope, Import	
Week 12	May 22	Turtles, Writing Classes	Assignment part 2 due 8:00am Tue, May 23
Week 13	May 29	Revision	
Mid Year Examination Period - MUST be available normal & supplementary period			

Lecture Topics and Lab topics are the same for each week

INFT1004

Visual Programming

Module 11.1
Making Movies

Guzdial & Ericson - Third Edition – chapter 13
Guzdial & Ericson - Fourth (Global) Edition – chapter 14

Frames

Movie/animations use a sequence of images in order to create an illusion of change.

Often the change involves some kind of movement in position or orientation of objects.

But other properties may also change such as.. colour, lighting, shape,....

Frames

The images need to change at a rate of at least 12 frames per second or changes may appear disjointed and the illusion broken.

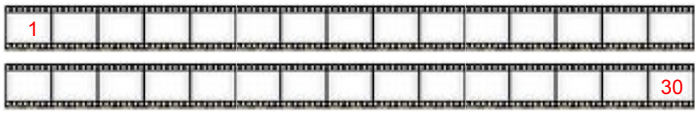
Typical Frame Rates

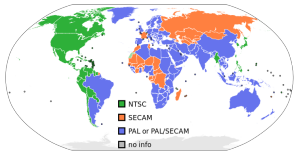
Hand-drawn animated characters typically use about 15 frames per second. It is cheaper to produce a lower number of hand-drawn frames and often realism is not the primary goal.

Old movies used 16 frames per second

TV Frame Rates

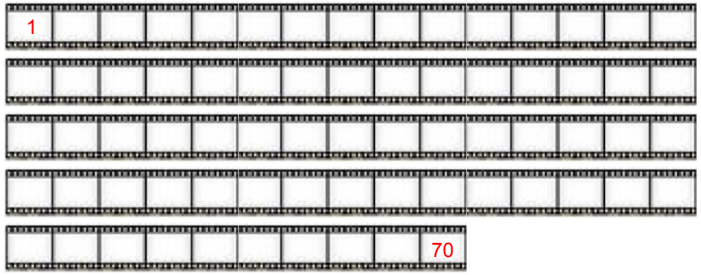
Television typically runs at 24-30 frames per second. The illusion of movement is more realistic with faster rates.





Fastest Frames

After about 70 frames per second there is no further improvement due to the physical limitations of human perception.



Examples - Movies

RedSquare – simple animation with draw functions

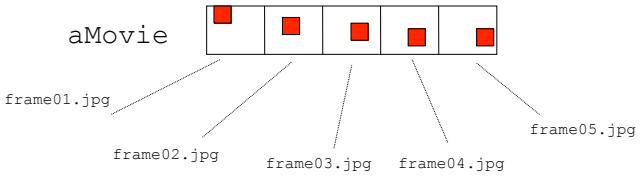
Ticker Tape – simple animation of text

Fading View – movie effect using pictures

A List of Frames / Pictures

It might be obvious that a movie/animation could be represented as a list of pictures (one per frame).

So how do we do this in JES?

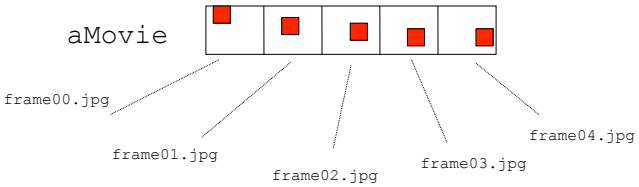


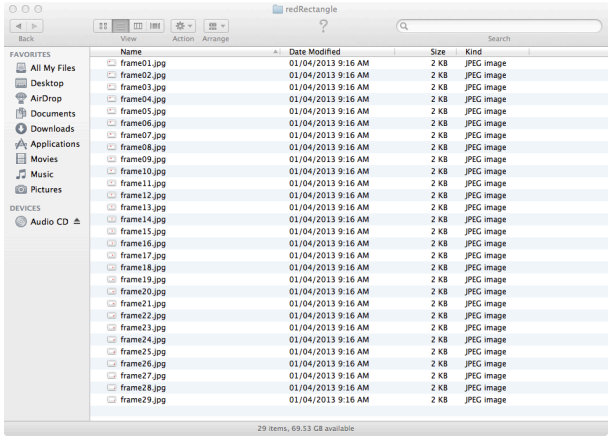
Iterating Frames

We represent frames as JPEG pictures.

One JPEG file per frame.

(When processing movies, we're going to create sequence of JPEG files.)

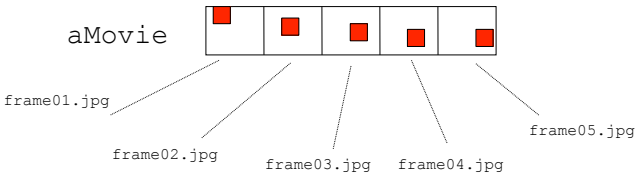




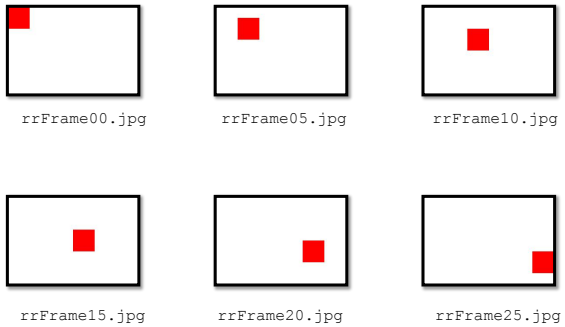
We specify the order by encoding the frame number into the name and putting all the frame images in the same directory

Frame Order

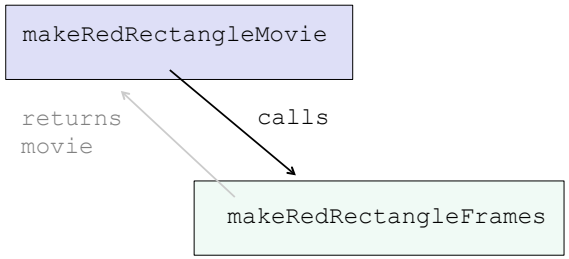
With leading zeroes eg. "frame01.jpg" every filename has the same length, the order is alphabetical as well as numerical.



Red Rectangle Movie



Red Rectangle Movie



Red Rectangle Movie

```
def makeRedRectangleMovie():
   # This function test the making of an avi movie/animation

   movie = makeRedRectangleFrames()
   playMovie(movie)

   #save movie to avi format
   movieFilename = getMediaPath() + "redRectangle.avi"

   print (movieFilename)

   writeAVI(movie, movieFilename)
```

Mod11_01_MakingMovies.py

Red Rectangle Movie

```
def makeRedRectangleFrames():
   directory = getMediaPath()

   #create 30 frames - from frame00.jpg to frame29.jpg
   for i in range(0,30):
      frame = makeEmptyPicture(300,200)
      addRectFilled(frame,i*10, i*5, 50,50, red)
      numberString=str(i)

      #set up the picture file name
      if i < 10:
         name = directory + "/rrFrame0" + numberString +".jpg"
         writePictureTo(canvas,name)
      else:
         name = directory + "/rrFrame" + numberString +".jpg”
         writePictureTo(canvas, name)

   #now make the movie
   movie=makeMovieFromInitialFile(directory + "/rrFrame00.jpg")
   return movie
```

Ticker Tape Movie

Ticker tape displays have some kind of scrolling text that moves across the screen

Lets try the message
`"Programming is very very cool!"`

(I used the requestString function so I can type in a different message anytime)

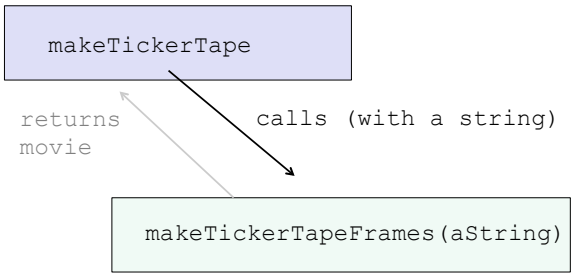
Mod11_01_MakingMovies.py

Ticker Tape Movie

ttFrame000.jpg ttFrame009.jpg ttFrame030.jpg

ttFrame060.jpg ttFrame90.jpg ttFrame119.jpg

Ticker Tape Movie



Ticker Tape Movie

```
def makeTickerTape():
   # This function test the making of an ticker tape movie (avi)
   # The message is entered by the user

   message=requestString("Enter the message to use in ticker tape"
   movie = makeTickerTapeFrames(message)
   #playMovie(movie)

   #save movie to avi format
   movieFilename = getMediaPath() + "tickerTape.avi"
   print (movieFilename)
   writeAVI(movie, movieFilename)  #save the avi file
```

Mod11_01_MakingMovies.py

Ticker Tape Movie

```
def makeTickerTapeFrames(aString):

   directory = getMediaPath()
   print("directory=" + directory)

   for n in range(0,120):
      frame = makeEmptyPicture(300,100)
      addText(frame, 300-(n*4), 50, aString)
      endString=str(n) + ".jpg"
      if n < 10:
         writePictureTo(frame, directory + "/ttFrame00" +endString)
      elif n < 100:
         writePictureTo(frame, directory + "/ttFrame0" +endString)
      else:
         writePictureTo(frame, directory + "/ttFrame" +endString)

   movie=makeMovieFromInitialFile(directory + "/ttFrame000.jpg")
   return movie
```


Fading View

Lets use our picture processing skills to create more of a movie after effect.

We will start with a picture and then create frames that fade to black

(we could use another colour such as white or red or blue – so lets try and write code that we can easily modify for different colours)

Mod11_01_MakingMovies.py

Goodbye Barbara – The Movie

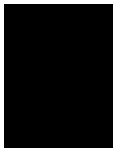

fpFrame00.jpg


fpFrame10.jpg

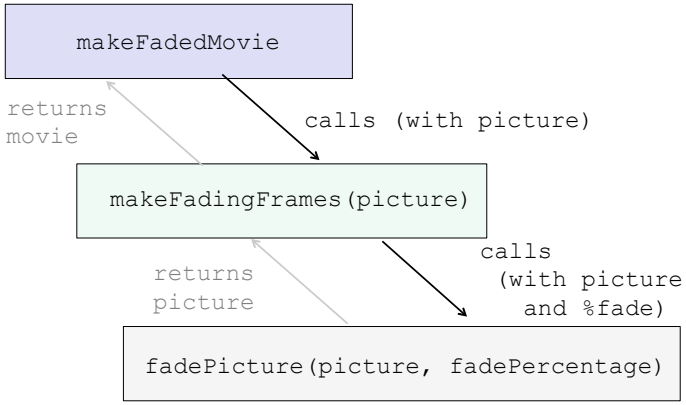

fpFrame20.jpg


fpFrame30.jpg


fpFrame40.jpg


fpFrame50.jpg

Fading Movie



Fading Movie

```
def makeFadedMovie():
   # This function test the making of a movie based on
   # an existing picture - by creating a fade to black

   file = pickAFile()
   picture = makePicture(file)
   movie = makeFadingFrames(picture)
   playMovie(movie)

   #save movie to avi format
   movieFilename = getMediaPath() + "FadedMovie.avi"
   print (movieFilename)
   writeAVI(movie, movieFilename)
```

Mod11_01_MakingMovies.py

Fading Movie

```
def makeFadingFrames(picture):
   directory = getMediaPath()

   for n in range(0,51):
      # this function creates the new frame - each one faded a
      # bit more - 50 frames - so starts at 0% end at 100%
      frame = fadePicture(picture,n*2)

      endString = str(n) + ".jpg"

      if n < 10:
         writePictureTo(frame, directory + "/fpFrame0" +endString)
      else:
         writePictureTo(frame, directory + "/fpFrame" +endString)

   movie=makeMovieFromInitialFile(directory + "/fpFrame00.jpg")

   return movie
```


Fading Movie

```
def fadePicture(picture, fadePercentage):

   colorAmount = fadePercentage / 100
   pictureAmount = (100 - fadePercentage) /100.0

   redAdd   = 0 * colorAmount
   greenAdd = 0 * colorAmount
   blueAdd  = 0 * colorAmount

   fadedPicture = duplicatePicture(picture)

   for pixel in getPixels(fadedPicture):
      newRed   = int((getRed(pixel) * pictureAmount) + redAdd)
      newGreen = int((getGreen(pixel) * pictureAmount) + greenAdd)
      newBlue  = int((getBlue(pixel) * pictureAmount) + blueAdd)
      setRed(pixel, newRed)
      setGreen(pixel, newGreen)
      setBlue(pixel, newBlue)

   return(fadedPicture)
```


INFT1004

Visual Programming

Module 11.2
Scope and Import

Guzdial & Ericson - Third Edition – chapter 10
Guzdial & Ericson - Fourth (Global) Edition – chapter 11

Argument and parameter names

Some programming novices get confused about the names of arguments and parameters

parameter

```
def upVolume(aSound):
```

Do we have to call it with

```
>>> upVolume (aSound)
```

argument

Argument and parameter names

Some programming novices get confused about the names of arguments and parameters

parameter

```
def upVolume(aSound):
```

Do we have to call it with

```
>>> upVolume (someSound)
```

argument

Answer: the function name `upVolume` has to be the same;

the argument and parameter can have different names.

Argument and parameter names

Remember, when defining a function the parameter is a placeholder name for whatever argument will be passed in.

parameter

```
def upVolume(aSound):
```

The parameter exists only within the function it's defined in.

If the same parameter name is used in several functions, Python treats it as several distinct names, one in each.

Argument/parameter assignment

The argument/parameter relationship is just like assignment – different for simple types and objects

If an argument is a simple type (int, float, string, etc), the parameter takes on the value of the argument when the function is called

Argument/parameter assignment

The argument/parameter relationship is just like assignment – different for simple types and objects

If an argument is a simple type (int, float, string, etc), the parameter takes on the value of the argument when the function is called

If an argument is an object (picture, pixel, sound, sample, file, etc), the parameter becomes another name for the argument when the function is called (a reference)

This is why, if a function modifies an object-type parameter such as a picture, the argument itself might also be modified (side effect)

Scope

Scope describes in which parts of the program a name exists. In general terms . . .

A name that is defined outside any function (eg in the command area) exists both in the command area and in all the functions of the program; its scope is 'global';

It's not good practice to use global variables within functions; functions should be self-contained.

There will never be a need to use global variables in your programs (or maybe ever as a programmer)

Scope

A name that is used as a parameter exists only within the function; its scope is 'local' to the function.

If a parameter name is the same as a global name, the global name suffers a 'hole' in its scope – it ceases to exist while the function is executing.

Scope

```
def testScope():

   file = pickAFile()
   file2 = pickAFile()
   picture = makePicture(file)
   picture2  = makePicture(file2)
   newPicture = emptyMe(picture2)

 def emptyMe(picture):

    width = getWidth(picture)
    height = getHeight(picture)

    newPicture = makeEmptyPicture(width, height)
    return newPicture
```

The variable called *picture* has no relationship to the *emptyMe* function's parameter called *picture*

They have different scopes.

Scope

```
def testScope():

   file = pickAFile()
   file2 = pickAFile()
   picture = makePicture(file)
   picture2  = makePicture(file2)
   newPicture = emptyMe(picture2)

 def emptyMe(picture):

    width = getWidth(picture)
    height = getHeight(picture)

    newPicture = makeEmptyPicture(width, height)
    return newPicture
```

The variable called *newPicture* has no relationship to the *emptyMe* variable called *newPicture*

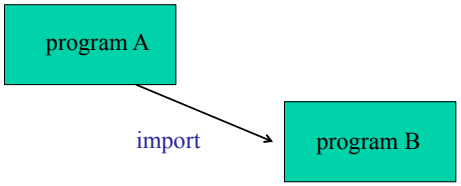
They have different scopes.

Of course after the return statement the *newPicture* in testScope just happens to be assigned the *newPicture* returned from *emptyMe*

Scope

What if you want to use functions you have in one program in another program?

Importing from other files



You can tell a program (B) to use functions written in another program (A).

Importing from other files

This is done by using `import`

which imports (loads) functions from the specified Python program file

Importing from other files

This is done by using `import`

which imports (loads) functions from the specified Python program file

The program file you're importing from must be in the library path, which you set using the JES function `setLibPath()`

Not Importing from other files

If the functions you're writing for import include JES media features, you must put

```
from media import *
```

at the start of the file to be imported, to import these functions from media.py

Not Importing from other files

media.py is in Program Files / JES 4.3 / Sources, so you must `setLibPath()` to this folder

Therefore you must put your own import functions in this same folder, far from your other files

Not Importing from other files

media.py is in Program Files / JES 4.3 / Sources, so you must `setLibPath()` to this folder

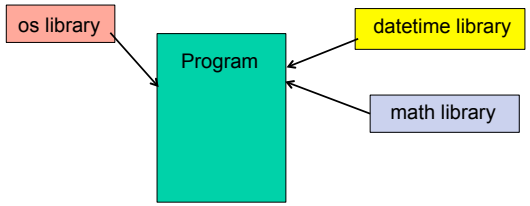
Therefore you must put your own import functions in this same folder, far from your other files

Fortunately, this doesn't matter right now because you must keep all your programs completely self-contained – or it won't be possible to mark your assignments – so you shouldn't be explicitly importing functions

Code in libraries

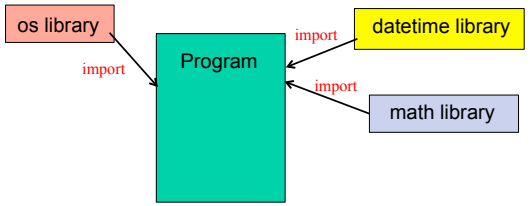
When we do media computation, we're using lots of code provided for us in JES 'libraries'

Most programming languages offer libraries of modules that the programmer can call on



Code in libraries

To use Python modules other than the JES ones we need to specifically request them – this is called importing them



Importing a module

```
import modulename
```

If we import a module name, to use a function in that module we need to write..

```
modulename.functionname()
```


Importing a module

<u>Example</u>: The operating system (os) module, which handles file paths and names

```
import os

print os.listdir("C:\university\Inft1004\PythonProgs”)
```

Importing a module

Example: The operating system (os) module, which handles file paths and names

```
import os

print os.listdir("C:\university\Inft1004\PythonProgs”)
```

If we wish (eg if the module has a long name) we can rename it on import:

```
import java.awt.event as event
```


Importing functions from a module

If we import specific functions from a module, we can access them without the module prefix:

```
from os import listdir

print listdir("C:\university\Inft1004\PythonProgs")
```


Importing functions from a module

We can even import all the functions from a module in this way, and access them without the module prefix:

```
from os import *

print listdir("C:\university\Inft1004\PythonProgs")
```


The *random* module

```
from random import *
```

Mod11_02 _Import.py

The *random* module

Random numbers, sometimes incredibly useful in programming, are from the module random

```
random.random()
```

– a quasi-random number in the range 0 to 1 (0 is included, 1 is excluded)

Mod11_02 _Import.py

The *random* module

Random numbers, sometimes incredibly useful in programming, are from the module random

```
random.random()
```

– a quasi-random number in the range 0 to 1 (0 is included, 1 is excluded)

Remember if you

```
from random import *
random()
```

Mod11_02 _Import.py

The *random* module

```
random.randint(start, end)
```

– a random integer between start and end inclusive

Mod11_02 _Import.py

The *random* module

```
random.choice(list)
```

– a randomly chosen element from list

Mod11_02 _Import.py

The *random* module

```
random.randrange(start, end, step)
```

– a random integer in the range from start to just short of end (with the specified step, if that's included)

Other useful modules

- datetime
- calendar
- math
- zipfile
- email
- SimpleHTTPServer

Other useful modules

- datetime
- calendar
- math
- zipfile
- email
- SimpleHTTPServer

If you want to find out more about any of these, how would you go about it?

The official Python documentation is freely available online