

Introduction to JES and Programming

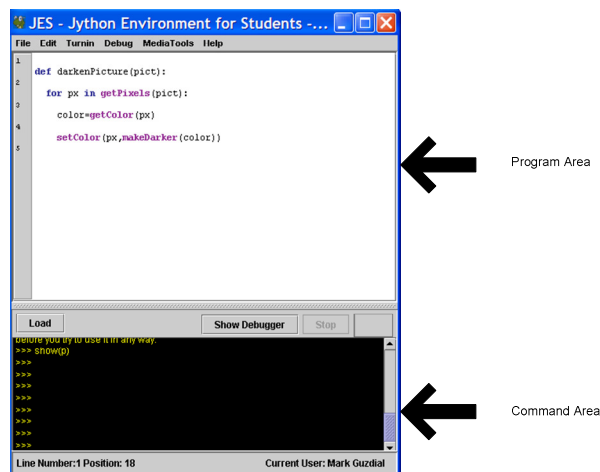
Installation

- Installing JES and starting it up
 - Windows users:
 - Just copy the folder
 - Double-click JES application
 - Mac users:
 - Just copy the folder
 - Double-click the JES application
- There is help available from the Help menu

We will program in JES

- JES: Jython Environment for Students
- A simple *editor* (for entering in our *programs* or *recipes*): We'll call that the *program area*
- A *command* area for entering in commands for Python to execute.

JES - Jython Environment for Students



Tour of JES

- Save and Save As
 - Save your files as “filename.py”
- Cut/Copy/Paste with shortcut keys
- Turning in assignments
 - Don’t use the Turnin feature of JES
 - Just send your code files to me in email as attachments
- Help
 - Explain is contextualized help: Highlight a JES (media) function
 - Lots of help on mediatools and the like

Python understands *commands*

- We can name data with **=**
- We can print values, expressions, anything with **print**

Using JES

```
>>> print 34 + 56
90
>>> print 34.1/46.5
0.7333333333333334
>>> print 22 * 33
726
>>> print 14 - 15
-1
>>> print "Hello"
Hello
>>> print "Hello" + "Mark"
HelloMark
```

Some Operators

- + addition
- - subtraction
- * multiplication
- / division
- % modulus (gives remainder after division)
- ** exponentiation

What will JES output?

```
>>> print 16 / 4 * 3
```

```
>>> print 10 % 3
```

```
>>> print 10 % 2
```

```
>>> print 57 % 25
```

```
>>> print 2 ** 3
```

What will JES output?

```
>>> print 4 / 3
```

← Evaluated as INTEGERS
Anything after the decimal point is thrown away

```
>>> print 7 / 4
```

← Turn numbers into FLOATING POINT
values to avoid; e.g. 4.0 / 3

```
>>> print 2 + 3 * 4 + 2
```

← * and / take **precedence** over + and -

```
>>> print 2 + 4 / 2 + 2
```

←

```
>>> print 2 / 4 * 2
```

← Left to right evaluation at same
level of precedence

Precedence Rules

- Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want
 - $(1+1)**(5-2)$ is 8.
 - Use parentheses to make an expression easier to read, as in $(2 + (3 * 4) - 2)$, even though it doesn't change the result.
- Exponentiation has the next highest precedence
 - $2**1+1$ is ?
 - $3*1**3$ is ?
- Multiplication, Division, and Modulus have the same and next highest precedence
- Addition and Subtraction have the same and next highest precedence
- Operators with the same precedence are evaluated from left to right
 - $3*100/60$, the multiplication happens first, yielding $300/60$, which in turn yields 5. If the operations had been evaluated from right to left, the result would have been $3*1$, which is 3

Command Area Editing

- Up/down arrows walk through *command history*
- You can edit the line at the bottom
 - and then hit Return/Enter
 - that makes that last line execute

Demonstrating JES for files

```
>>> print pickAFile()
C:\Documents and Settings\Kenrick\My
Documents\Class\CSA109\guzdial_python\content\MediaSources\arthurs-seat.jpg

>>> print makePicture(pickAFile())
Picture, filename C:\Documents and Settings\Kenrick\My
Documents\Class\CSA109\guzdial_python\content\MediaSources\arch.jpg height 480
width 360

>>> myfilename = pickAFile()
>>> print myfilename
C:\Documents and Settings\Kenrick\My
Documents\Class\CSA109\guzdial_python\content\MediaSources\arch.jpg

>>> mypicture = makePicture(myfilename)
>>> print mypicture
Picture, filename C:\Documents and Settings\Kenrick\My
Documents\Class\CSA109\guzdial_python\content\MediaSources\arch.jpg height 480
width 360
>>> show(mypicture)
```

Demonstrating JES for sound

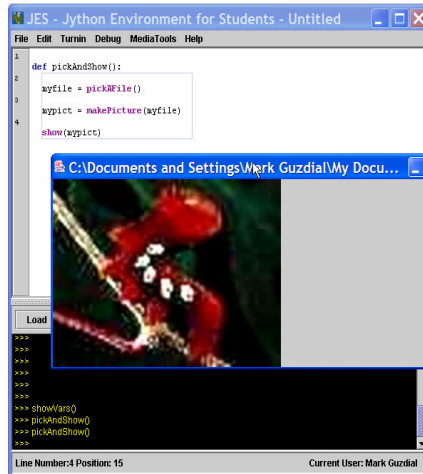
```
>>> print pickAFile()
C:\Documents and Settings\Kenrick\My
Documents\Class\CSA109\guzdial_python\
content\MediaSources\aaah.wav

>>> print makeSound(pickAFile())
Sound of length 43009

>>> print play(makeSound(pickAFile()))
None
```

Writing a recipe: Making our own functions

- To make a function, use the command **def**
- Then, the name of the function, and the names of the input values between parentheses (“(input1)”)
- End the line with a colon (“:”)
- The *body* of the recipe is indented (Hint: Use two spaces)
 - That’s called a *block*



Making functions the easy way

- Get something working by typing commands
- Enter the **def** command.
- Copy-paste the right commands up into the recipe

A recipe for playing picked sound files

```
def pickAndPlay():  
    myfile = pickAFile()  
    mysound = makeSound(myfile)  
    play(mysound)
```

Note: **myfile** and **mysound**, inside **pickAndPlay()**, are *completely different* from the same names in the command area.

These are called **local variables**; variables used in different blocks are considered different, even if using the same name, if they are in different blocks.

Blocking is indicated for you in JES

- Statements that are indented the same, are in the same block.
- Statements that are in the same block as where the line where the cursor is are enclosed in a blue box.



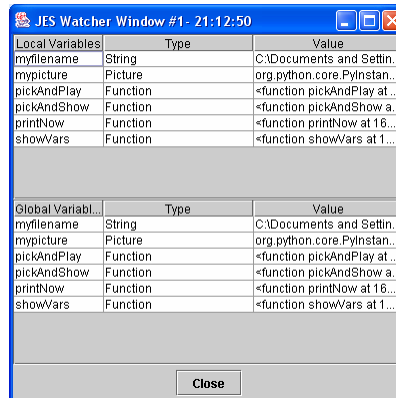
A function for playing picked picture files

```
def pickAndShow():  
    myfile = pickAFile()  
    mypict = makePicture(myfile)  
    show(mypict)
```

The Most Common JES Bug: Forgetting to Load

- Your function does **NOT** exist for JES until you *load* it
 - Before you load it, the program is just a bunch of characters.
 - Loading *encodes* it as an executable function
- Save and Save As
 - You must Save before Loading
 - You must Load before you can use your function

What if you forget your variable names? `showVars()`



MOST IMPORTANT THING TO DO TO PASS THIS CLASS!

- *DO THE EXAMPLES!*
- Try them out for yourself. Try to replicate them. *Understand them*
 - EVERY CLASS, TYPE IN AT LEAST *TWO* OF THE EXAMPLES FROM CLASS
- To understand a program means that you know why each line is there.
- What not to do: try changing the program “randomly” until it hopefully works
- You will encounter all the simple-but-confusing errors *early*—**BEFORE** you are rushing to get homework done!!

All about naming

- We name our data
 - Data: The “numbers” or values we manipulate
 - The names for data are “*variables*”
- We name our recipes/functions
- Quality of names determined much as in Philosophy or Math
 - Enough words to describe what you need to describe
 - Understandable
 - E.g., don’t use **interestRate** to store account balance

Naming our Encodings

- We even name our encodings (something is a number, something else is text...)
 - Sometimes referred to as *types*
- Some programming languages are *strongly typed*
 - A name has to be *declared* to have a type, before any data is associated with it
 - Python is not strongly typed

```
>>> x = 3
>>> print x
3
>>> x = "hello"
>>> print x
hello
>>>
```

Programs contain a variety of names

- You will name your *functions*
 - Just like functions you knew in math, like sine and gcd (Greatest Common Divisor)
- You will name your *data (variables)*
- You will name the data that your functions work on
 - *parameters*, like the 90 in sine(90)
- Key: Names inside a function only have meaning while the function is being executed by the computer.

Names for things that are *not* in memory

- A common name that you'll deal with is a *file name*
 - The program that deals with those is called the *operating system*, like Windows, MacOS, Linux
- A file is a collection of bytes, with a name, that resides on some external medium, like a *hard disk*.
 - Think of it as a whole bunch of space where you can put your bytes (your information)
- Files are typed, typically with three letter *extensions*
 - .jpg files are JPEG (pictures), .wav are WAV (sounds)

Names can be (nearly) anything

- Must start with a letter (but can *contain* numerals or _)
- Can't contain spaces or other punctuation
 - `myPicture` is okay but `my Picture` is not
- Be careful not to use command names as your own names
 - `print` = 1 won't work
 - (Avoid names that appear in the editor pane of JES highlighted in blue or purple)
- *Case matters*
 - `MyPicture` is not the same as `myPicture` or `mypicture`
- Sensible names are sensible
 - E.g. `myPicture` is a good name for a picture, but not for a sound file.
 - `x` could be a good name for an x-coordinate in a picture, but probably not for anything else - it is too vague

JES Functions

- Many functions are pre-defined in JES for sound and picture manipulations
 - `pickAFile()`
 - `makePicture()`
 - `makeSound()`
 - `show()`
 - `play()`
- Some of these functions accept *input* values called **parameters** or **arguments**

```
theFile = pickAFile()  
pic = makePicture(theFile)
```

Picture Functions

- **makePicture**(filename)
creates and returns a picture object, from the JPEG file at the filename
- **show**(pictureObject)
displays a picture object in a window
- We'll learn functions for manipulating pictures later, like **getColor**, **setColor**, and **repaint**

Sound Functions

- **makeSound**(filename)
creates and returns a sound object, from the WAV file at the filename
- **play**(sound)
plays the sound
 - but doesn't wait until it's done
 - blockingPlay**(sound) waits for the sound to finish
- We'll learn more later like **getSample** and **setSample**

A value come from: the value itself, a variable name that holds that value, a function that returns the value

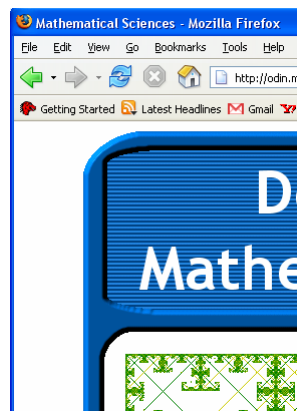
```
>>> file=pickAFile()
>>> print file
C:\Documents and Settings\Kenrick\My
Documents\Class\CSA109\guzdial_python\content\MediaSources\ar
thurs-seat.jpg
>>> show(makePicture(file))
>>> show(makePicture(r"C:\Documents and Settings\Kenrick\My
Documents\Class\CSA109\guzdial_python\content\MediaSources\ar
thurs-seat.jpg"))
>>> show(makePicture(pickAFile()))
```

**Put r in front of Windows filenames:
r"C:\mediasources\pic.jpg"**

Grabbing media from the Web

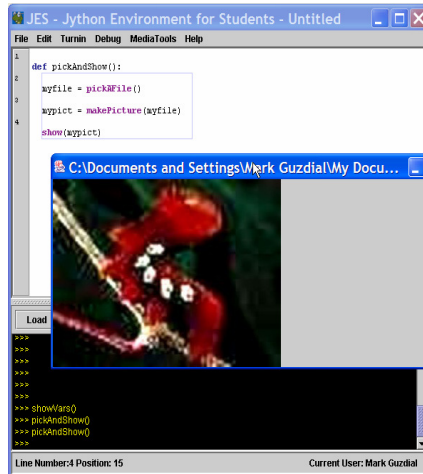
- Right-click (Windows) or Control-Click (Mac)
- Save Target As...
- Can *only* do JPEG images (.jpg, .jpeg)

Most images on the Internet are copyright. Without permission you can download and use them for your use *only*.



Writing a recipe: Making our own functions

- To make a function, use the command **def**
- Then, the name of the function, and the names of the input values between parentheses (“(input1)”)
- End the line with a colon (“:”)
- The *body* of the recipe is indented (Hint: Use two spaces)
 - That’s called a *block*



A recipe for playing picked sound files

```
def pickAndPlay():  
    myfile = pickAFile()  
    mysound = makeSound(myfile)  
    play(mysound)
```

Bug alert!!!

myfile and **mysound**, inside pickAndPlay(), are *completely different* from the same names in the command area.

Bug Example

```
>>> >>> myfile = pickAFile()
>>> print myfile
C:\Documents and Settings\Kenrick\My
Documents\Class\CSA109\guzdial_python\content\MediaSources\arch.jpg

def bugExample():
    mysound = makeSound(myfile)          ← There is no myfile defined!
    play(mysound)

>>> bugExample()
A local or global name could not be found. You need to define the function or
variable before you try to use it in any way.
Please check line 2 of C:\Documents and Settings\Kenrick\My
Documents\Class\CSA109\spring2006\samplepython\test.py

def bugExample():
    print myfile
```

A recipe for showing picked picture files

```
def pickAndShow():
    myfile = pickAFile()
    mypicture = makePicture(myfile)
    show(mypicture)
```

“Hard-coding” for a *specific* sound or picture

```
def playSound():  
    myfile = r"C:\bark.wav"  
    mysound = makeSound(myfile)  
    play(mysound)
```

```
def showPicture():  
    myfile = r"C:\boat.jpg"  
    mypict = makePicture(myfile)  
    show(mypict)
```

You can always replace data (a *string* of characters, a number.... whatever) with a name (*variable*) that holds that data

.... or vice versa.

Q: This works, but can you see the disadvantage?



Function parameters allow flexibility

```
def playNamed(myfile):  
    mysound = makeSound(myfile)  
    play(mysound)
```

```
def showNamed(myfile):  
    mypict = makePicture(myfile)  
    show(mypict)
```

Q: What functions do you need?

Q: What (if any) should be their parameter(s)?

A: In general, have enough functions to do what you want, easily, understandably, and flexibly

(try for more generic, less specific functions)

Multiple Parameters

- Separate by a comma

```
def showAndPlay(mypicturefile, myaudiofile):  
    mypict = makePicture(mypicturefile)  
    show(mypict)  
    mysound = makeSound(myaudiofile)  
    play(mysound)
```

```
>>> pictfile = pickAFile()  
>>> soundfile = pickAFile()  
>>> showAndPlay(pictfile, soundfile)
```

What can go wrong when things look right?

- Did you use the *exact* same names (case, spelling)?
- All the lines in the block must be *indented*, and *indented the **same** amount*.
- Variables in the command area don't exist in your functions, and variables in your functions don't exist in the command area.
- The computer can't read your mind.
 - It will only do exactly what you tell it to do.

Programming is a craft

- You don't learn to write, paint, or ride a bike by attending biking lectures and watching others bike.
 - You learn to bike by *biking*!
- Programming is much the same.
 - You have to try it, make many mistakes, learn how to control the computer, learn how to *think* in Python.
- *The programming and labs that you have to write in this class aren't enough!*
 - *Do programming on your own!*
 - *Play around with the class and book examples!*