



# COMP2720: Automating Tools for New Media

Introduction to Programming &  
Jython Environment for Students



# Much of programming is about naming

- We name our data
  - Data: The “numbers” we manipulate
  - We call our names for data **variables**
- We name our recipes/programs
  - Do not use names like *myprogram1*, *myprogram2*
- Quality of names determined much as in Philosophy or Math
  - Enough words to describe what you need to describe
  - Understandable



# Naming our Encodings

- We even name our encodings
  - Sometimes referred to as **types**
  - For example, integer numbers, characters, etc.
- 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



# Our programs work with a variety of names

- You will name your functions
  - Just like functions you knew in math, like *sin* (*sine trigonometric function*) and *gcd* (Greatest Common Divisor arithmetic function)
- You will name your data (variables)
- You will name the data that your functions work on
  - Inputs, like the *90* in *sine(90)*
- Key concept: Names inside a function only have meaning while the function is being executed by the computer. (You'll see what we mean.)



# 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
- Files are typed, typically with three letter extensions
  - “.jpg” files are JPEG (pictures), “.wav” are WAV (sounds)

# Names can be (nearly) whatever we want

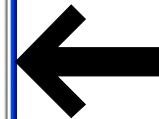
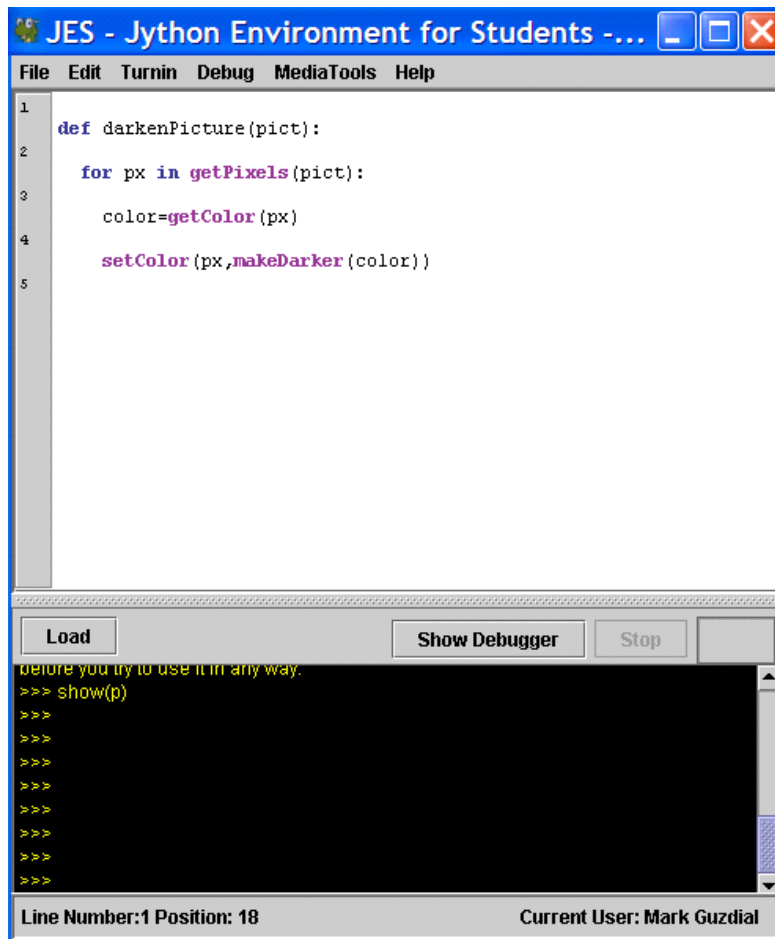
- Must start with a letter (but can contain numerals)
- Cannot contain spaces
  - myPicture is okay but my Picture is not
- Be careful not to use command names as your own names
  - print = 1 won't work
  - Avoid names that appear in the editor pane of JES highlighted in blue or purple
- Case matters
  - 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 picture file.
  - x could be a good name for an x-coordinate in a picture, but probably not for anything else



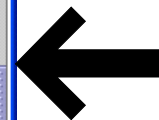
# We will program in JES

- JES: **J**ython **E**nvironment for **S**tudents
- 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



Program Area



Command Area



# Tour of JES

- Save and Save As
- Cut/Copy/Paste with shortcut keys
- Turnin (we will not need this!)
- Help
  - Explain is contextualised help: Highlight a JES function
- Command area editing
  - Up/down arrows walk through command history
  - You can edit the line at the bottom (just put the cursor at the end of the line before hitting Return/Enter).



# Python understands commands

- We can name data with =
- We can print values, expressions, anything with print
- Appendix A in text book: *Quick reference to Python*

# Using JES

```
>>> print 34 + 56 #this part is a comment
90
>>> print 34.1/46.5 #comments are ignored
0.73333333333333333334
>>> print 22 * 33
726
>>> print 14 - 15
-1
>>> print "Hello"
Hello
>>> print "Hello" + "Alexei"
HelloAlexei
```

# JES functions

- A bunch of functions are pre-defined in JES for sound and picture manipulations
  - pickAFile()
  - makePicture()
  - makeSound()
  - show()
  - play()
- Some of these functions accept input values

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



# Picture functions

- `makePicture(filename)` creates and returns a picture object, from the JPEG file at the filename
- `show(picture)` displays a picture 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)` makes the sound play
  - 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()`

# Completely the same: Values, names for those values, functions that return those values

```
>>> file=pickAFile()
```

```
>>> print file
```

```
C:\Documents and Settings\Mark Guzdial\My Documents\mediasources\barbara.jpg
```

```
>>> show(makePicture(file))
```

```
>>> show(makePicture(r"C:\Documents and Settings\Mark Guzdial\My  
Documents\mediasources\barbara.jpg"))
```

```
>>> show(makePicture(pickAFile()))
```

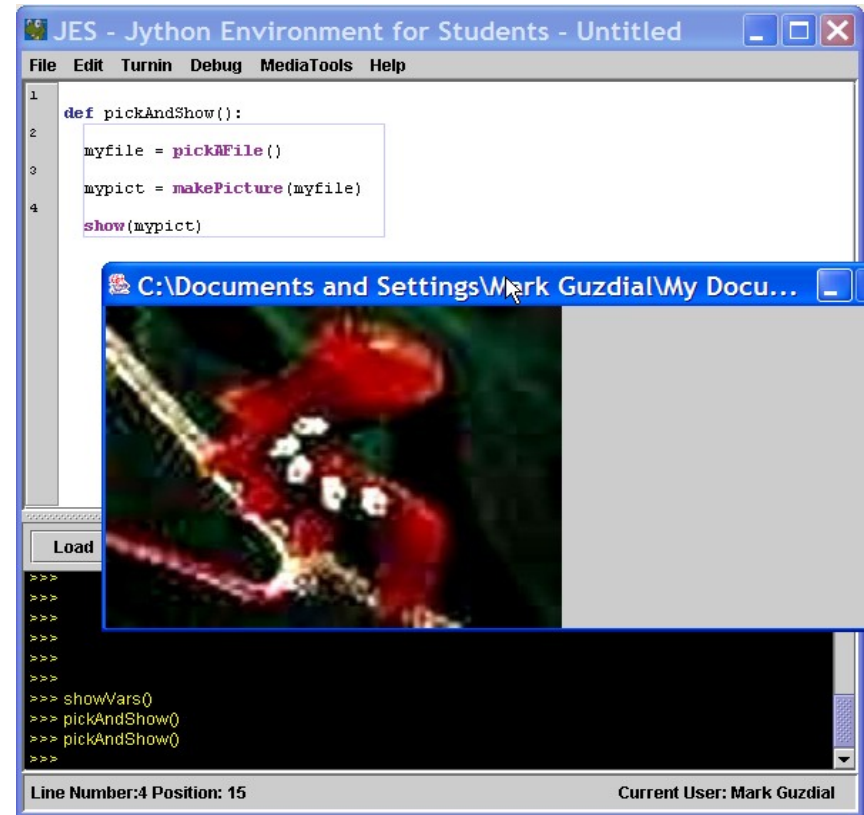
**Put 'r' in front of Windows filenames:  
r"C:\mediasources\pic.jpg"**

# Demonstrating JES for files and sounds

```
>>> print pickAFile()
/Users/guzdial/mediasources/barbara.jpg
>>> print makePicture(pickAFile())
Picture, filename /Users/guzdial/mediasources/barbara.jpg height 294 width 222
>>> print pickAFile()
/Users/guzdial/mediasources/hello.wav
>>> print makeSound(pickAFile())
Sound of length 54757
>>> print play(makeSound(pickAFile()))
None
>>> myfilename = pickAFile()
>>> print myfilename
/Users/guzdial/mediasources/barbara.jpg
>>> mypicture = makePicture(myfilename)
>>> print mypicture
Picture, filename /Users/guzdial/mediasources/barbara.jpg height 294 width 222
>>> show(mypicture)
```

# 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, e.g. `(myfile)`
- End the line with a colon (`:`)
- The body of the recipe is indented (Hint: Use two spaces)
  - That's called a block



```
JES - Jython Environment for Students - Untitled
File Edit Turnin Debug MediaTools Help
1
2 def pickAndShow():
3     myfile = pickAFile()
4     mypict = makePicture(myfile)
5     show(mypict)
Load
>>>
>>>
>>>
>>>
>>>
>>>
>>> showWars()
>>> pickAndShow()
>>> pickAndShow()
>>>
Line Number: 4 Position: 15 Current User: Mark Guzdial
```



# 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.

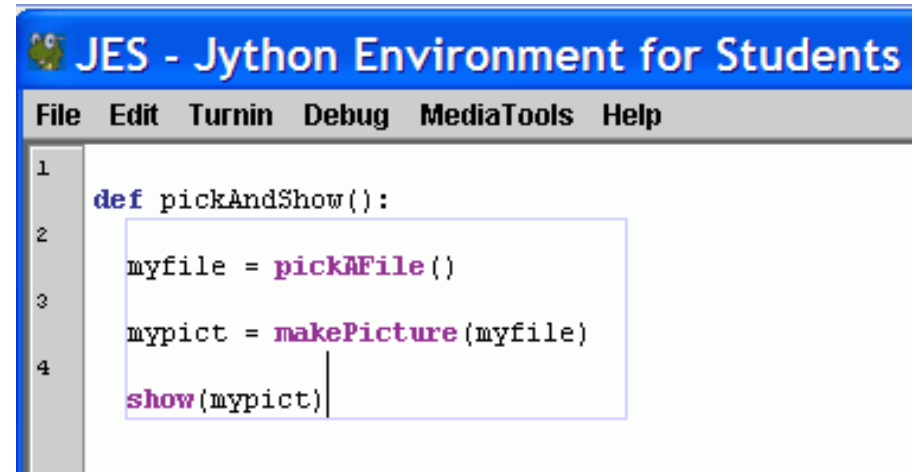


# **A function for showing picked picture files**

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

# 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.
- In *C*, *C++* or *Java*, `{}` are used to mark blocks.



```
JES - Jython Environment for Students
File Edit Turnin Debug MediaTools Help
1
2 def pickAndShow():
3     myfile = pickAFile()
4     mypict = makePicture(myfile)
5     show(mypict)
```

# What if you forget your variable names?

**Use:** `showVars()`



The screenshot shows a window titled "JES Debug Window #1 - 10:12:43". It contains two tables: "Local Variables" and "Global Variables". Both tables list the same variables: myfile (String), mypicture (Picture), printNow (Function), and showVars (Function). The values for myfile and mypicture are truncated in the image. Below the tables is a "Close" button. At the bottom of the window, a command prompt shows the following code:

```
>>> myfile=pickAFile()
>>> mypicture=makePicture(myfile)
>>> showVars()
>>>
```



# 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



# Programming is a craft

- You don't learn to paint, or knit by attending painting or knitting lectures and watching others paint or knit.
  - You learn to paint or knit by doing it.
- 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 programs that I have showed you in this lecture are not enough!
  - Do programming on your own!



# Things to do after lectures

- DO THE EXAMPLES! (after every lecture)
- Try them out for yourself. Try to replicate them, and try to understand them
  - Every week, type in at least TWO of the examples from the lectures
- To understand a program means that you know why each line is there.
- You will encounter all the simple-but-confusing errors early — BEFORE you are rushing to get labs, home works or assignments done!!
- Please read chapter 2 in text book by the end of week 2