A crash course in programming in Python

Steve Kautz
What are we doing here?

• A) What am I doing here?
• B) What are you doing here?
What am I doing here?
What am I doing here?

– Hridesh!
– I am so gullible...
What are you doing here?

• Cover a 15-week course in 3 hours?
“Stick them with the pointy end”

See: [http://norvig.com/21-days.html](http://norvig.com/21-days.html)
Organization?

- Concentrate on big picture, trade secrets
- Don’t sweat the small stuff, fill in details later according to time, interest, necessity
Resources

• See
http://web.cs.iastate.edu/~smkautz/bigdata/resources.html

• Will steal some material from There Are Eels In My Hovercraft (Com S 104)

• Stay tuned...

• As time permits, include some hands-on exercises
Organization?

• **8:30 – 10:30**
  – Programming and Python overview
  – Control structures (conditionals, loops)
  – Defining functions and “modules”, using libraries
  – Basic string operations, indexing and slices

• **11:00 – 12:30**
  – Structured types: Lists and dictionaries
  – Reading files
  – Interacting with the OS and “environment”
I don’t want to be a programmer

• Don’t need to be an expert to do useful things!
  – Write programs or “scripts” whose main purpose is to invoke other programs
  – Read files, pre-process data
I don’t want to be a programmer

• I’m not a “Python expert”
  – Learned Python to write scripts for some data processing tasks
  – Later, used for teaching intro courses

• The beauty of Python is how much you can accomplish *without* getting bogged down in language details!
I don’t want to be a programmer

• Forget about programming for a minute
• How do you answer simple questions about data?
Example: find the largest number in a list

43 17 67 32 86 79 18
Example: find the largest number in a list

Example: find the largest number in a list

196 61 428 522 398 144 274 23 354 145 47 511 516 278 507 526 404 27 16 486 320
284 56 395 318 178 4 217 354 160 279 510 98 345 568 257 98 435 234 493 124 171
96 542 397 349 393 540 379 37 513 159 329 213 58 404 526 170 33 422 399 535 323
150 432 422 116 231 140 434 188 305 542 449 504 108 99 410 189 259 83 573 310
332 51 120 244 325 530 417 84 180 407 391 93 264 484 570 530 17 102 324 76 248
564 265 247 170 262 370 509 108 398 176 370 21 527 61 544 517 495 49 29 185 198
395 112 77 37 417 350 83 444 149 469 299 346 281 512 351 95 474 443 488 517 83
494 18 330 438 336 49 371 32 402 241 442 549 64 43 338 206 571 246 519 18 284
513 133 205 305 481 90 518 297 565 184 210 131 270 238 24 532 142 168 28 1 364
128 90 355 277 42 543 440 223 546 193 245 490 270 218 226 259 442 392 481
316 185 395 437 331 148 48 34 21 407 14 109 312 274 350 372 516 445 566 354
247 8 196 134 501 275 271 190 91 298 23 506 511 313 337 40
55 384 227 525 103 18 52 378 436 519 371 164
55
354 515 37 166 317 351 213 499 296 30 282 156
361 465 25 103 18 34 305 19 445 374 379 486 112 522 392 388 287
“Pseudocode”

write down the first number
for each number in the list
  if it’s bigger than the one we have,
    write that one down instead
return the last one written down
Our programming model

A picture of what a computer can do:

A reasonably bright 4th-grader

Clipboard

Pocket calculator
Basic ingredients of computation

1) Store a value so we can remember it later
2) Do basic arithmetic
3) Check a condition and take some action, depending on whether the condition is true
4) Repeat some action, continuing as long as a condition is true
5) Get input, produce output

That’s all we need!
That’s not *really* programming!

• Actually, we’ve done the hardest part
  – Identify the problem to be solved and criteria for correctness
    • Test case: given the list [43, 17, 67, 32, 86, 79, 18] return the answer 86
  – Write down steps are of a *strategy* or *algorithm*

• You would be amazed...
• To use a real computing machine, we just need to choose a formal programming language that the machine can interpret
Python!

```python
max = my_list[0]
for num in my_list:
    if num > max:
        max = num
print(max)
```

• Begs the question of what is “my_list” and how do we put our data into it... stay tuned
• Prints result on the screen ...?
Keywords (def, for, if, return, ...)
Operators (+, *, <, ...)
Literal values (42, 3.14, “Hello”) Identifiers (max, num, ...)
Syntax rules (grammar and punctuation)

(To use a language effectively, we also have to be familiar with its libraries - predefined collections of functions and data types.)
Python!

```python
max = my_list[0]
for num in my_list:
    if num > max:
        max = num

print(max)
```

- Unusual syntax rule: *indentation matters!*
Creating and running a program (aka script, module)

• Python programs are normally run by an “interpreter”
  – Essentially just reads and executes one line at a time
  – Syntax errors may not be discovered until runtime

• Traditional programming languages are first “compiled” (translated completely to machine code)

• Distinction is somewhat blurry (e.g. Java)
Options for creating and running a Python program or script

• Use an IDE (“integrated development environment”) such as IDLE or Wing to edit/run
  – IDEs may offer project management features, interactive debugger, etc
• Use any text editor, run the program in a command shell
• Use iPython and/or Jupyter
• Also can use Python Tutor for experimentation
Hello, World!

• Executes statements “in order”
  -- “Flow” can be altered by loops, conditionals, function calls
  -- `print(...)` is a built-in function in Python 3
    • Is a `keyword` in Python 2, used without parens
Try out some things in the interactive Python “shell”

• Expressions and types
• Arithmetic operators +, *, -, /, //, %
  Note / vs // in Python 3
• Boolean expressions
  – Relational operators >, <, >=, <=, ==, !=
  – Boolean operators and, or, not
• Variables and the assignment operator =
Try out some things in the shell

• Calling built-in functions
  – Value-returning functions: len(), pow(), max()
  – Functions with side effects: print()
    • print vs print() in Python 3

• Most functions are in modules that need to be imported
  – E.g. math.sqrt()

• Documentation and help(…)

Steve Kautz
Run our code

• We can create a list of literal values in Python by putting them in square brackets
  – Creates an instance of a type called “list” that contains a growable sequence of values

```python
my_list = [43, 17, 67, 32, 86, 79, 18]
max = my_list[0]
for num in my_list:
    if num > max:
        max = num
print("The max is", max)
```
Big picture moment

• What do I want the code to do, and how can I verify it?
  – Write simple test cases or usage examples

• Solve the problem by hand, describe the steps in your native language or pseudocode
  – Then translate into a formal language

• We can go home now!
Better organization

def find_max(my_list):
    max = my_list[0]
    for num in my_list:
        if num > max:
            max = num
    return max

- Defines a *procedure* or *function* called *find_max*
Better organization

• Put the `find_max` function in a separate module, say, `mystuff.py`
• Add an import statement and “qualify” the name

```python
import mystuff

test_list = [43, 17, 67, 32, 86, 79, 18]
m = mystuff.find_max(test_list)
print("The max is", m)
```
Big picture moment

• Organize work into sub-tasks that can be written as separate functions ("procedural decomposition")
  – Solve the problem incrementally
  – If they are useful, put them in a separate module
Reinventing the wheel

• `find_max(...)` is a nice example, but...
• actually, there is already a built-in function `max(...)` for finding the maximum value in a list

• A key skill is identifying libraries that already exist to do what you want, and *integrating* them into your application
Where are my files?

• Python interpreter works within an “environment” on your OS
  – Where does the shell look for your scripts?
  – Where does the shell look for imported modules?
Working directory

• A shell always has a “current working directory”

• If your script is in a different directory, need to
  – Provide a *path* to the file, OR
  – Change the working directory
    • Use `os.chdir()` in the Python shell
    • Use `%cwd` in Jupyter

• The import mechanism is a bit more tricky...
Another big picture moment

• Know where your files are!
Expressions vs statements

• An *expression* is a value, doesn’t “do” anything
  – \( 2 + 3, \text{“Hello”} \)
  – Call to value-returning function, \texttt{math.sqrt(25)}
  – Like a noun (e.g. “cat”)

• A *statement* is an instruction to DO something
  – \( x = 2 + 3 \)
  – Call to function with side-effect, \texttt{print(“Hello”)}
  – Like a sentence (e.g. “Please feed the cat!”)
More about defining functions

• Is it a value-returning function, or are we invoking it for its side-effects (e.g. producing output)?
  – Most functions are value-returning
  – Try to separate program logic from user interface and I/O
    • Code more likely to be reusable

• (E.g. see Chapters 7 and 8)
More about conditionals

• (See Chapters 5 and 6)
Indexing and slices

• (See Chapter 13)
String methods

- Functions vs “methods”
- `strip()`, `startswith()`, `endswith()`, `find()`
- Strings are immutable!
- Using the `in` keyword
- `split(...)"
More list operations

• Empty list []
• append(...), remove(...), index(...), del(...)
• .sort()
• Lists are references!
Dictionaries

- A list associates a value with an integer index
- A dictionary associates a value with an arbitrary key
  - Keys may be any immutable type

```python
votes = {}  # an empty dictionary
votes["Clinton"] = 17  # associate a value
votes["Perot"] = 6
votes["McCain"] = 2

for name in votes.keys():
    print(name)
    print(votes[name])
```
Sets

• A set is just a dictionary in which you only care about whether the keys are present, no values.

• Dictionaries and sets are designed so that lookups are efficient (generally independent of the size of the collection)

• Example – find elements of one list that also occur in a second list
Reading text files

• See Chapter 19...
Album of big picture moments

• Start with simple test cases or usage examples.
  – What do I want the code to do, and how can I verify it?
• Solve the problem by hand, describe the steps in your native language or pseudocode
• Work incrementally
  – Solve part of the problem, solve a simpler version of the problem
• Procedural decomposition: organize work into sub-tasks that can be written as separate functions (and tested independently)
  – If they are useful, put them in a separate module
• Whatever you’re trying to do, it might already be in a library somewhere
  – Often the “hard part” is just integrating modules that exist already
• Know where your files are
• Lists and dictionaries are your friends
Thank you!