What is Python?

Python is an interpreted programming language that allows you to do almost anything possible with a compiled language (C/C++/Fortran) without requiring all the complexity.

Main Features

- Automatic garbage collection
- Interpreted and interactive
- Object-oriented
- Useful built-in types
- Easy matrix algebra (via numpy)
- Easy to program GUIs
- Lot of documentation, tutorials and libraries

A Sample of Code ...

```
x = 4 - 1.0   # comment: integer difference
y = "Hello"   # double quotes
y = 'Hello'   # single quotes also work
print y, ' number ', x

if x == 0 or y == "Hello":
    x = x + 1
    y = y + " World" # concatenating two strings
print(y)
print(y * 3)  # repeating a string
len(y)        # String length
```

Language Introduction

- Assignment uses = and comparison uses ==
- + – * / % compute numbers as expected
- Use + for string concatenation
- Use % for string formatting (follows C conventions)
- Logical operators are words (and, or, not), but not symbols (&&, ||, !)
- First assignment to a variable will create it
- Python assigns the variable types

Words on Code Style

- Use consistent indentation (4 spaces) to mark blocks of code
- Use a newline to end a line of code or use \ when must go to next line prematurely)
- Comments start with # - the rest of line is ignored
- A *documentation string* can be included as the first line of any function or class with triple double-quotes
- Official style guide: PEP-8

Basic Data Types

- Integers (default for numbers)
- Strings
  - Can use " " or ‘ ’
  - Unmatched quotes can occur in the string: `"matt's"`
  - Use triple double-quotes for multi-line strings or strings which contain both ’ and " inside: `"""a'b"c"""`
- *Dynamic Typing* (Python determines data types automatically),
  - But Python is not casual about types, it enforces them thereafter: *Strong Typing*
  - e.g., you can't just append an integer to a string.

Naming Rules

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores (underscores have special meaning)

  `nao Nao _nao _2_nao nao_2 NAO`

- There are some reserved words:

  `and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while`

- Avoid overloading built-in functions:
  `len, type, abs, input,...`

List Objects

- List creation with brackets

```
lst = [10,11,12,13,14]
```

- Concatenating list

```
[10, 11] + [12, 13] # simply use the + operator
```

- Repeating elements in lists

```
[10, 11] * 2 # produces [10, 11, 10, 11]
```

- range(start, stop, step)

```
range(5)       # [0, 1, 2, 3, 4]
range(2,7)     # [2, 3, 4, 5, 6]
range(2,7,2)   # [2, 4, 6]
```

Indexing

- Retrieving and element

```
lst = [10, 11, 12, 13, 14]
lst[0]          # produces 10
```

- Setting and element

```
lst[1] = 21   # produces [10, 21, 12, 13, 14]
```

- Out of bounds

```
lst[10]         # raises and error
```

- negative indices count backward from the end of the list

```
lst[-1]         # produces 14
```

Assignment

- Multiple Assignment

```
x, y, z = 1, 2, 3 # y = 2
```

- Assignment creates object references

```
a = [0,1,2]
b = a          # x and y point at the same list
b[1] = 6       # changes to y also change x
print a
print b
b = [3, 4]     # re-assigning b to a new list
               # decouples the two lists
```

If Statements

- if/elif/else provide conditional execution of code blocks

```
x = 10
if x > 0:
    print 1
elif x == 0:
    print 0
else:
    print -1
```

- elif and else are not mandatory
- True means any non-zero number or non-empty object
- False means not true: zero, empty object, or `None`

For Loops

For loops iterate over a sequence of objects.

```
for i in range(5):
    print i,  # use , to suppress line break
# produces 0 1 2 3 4

for i in 'abcde':
    print i,
# produces  a b c d e

lst=['dogs','cats','bears']
for item in lst:
    print item + ' ',
# produces  dogs cats bears
```

While Loops

While loops iterate until a condition is met.

```
lst = range(3)
while lst:
    print lst
    lst = lst[1:]

# produces
# [0, 1, 2]
# [1, 2]
# [2]
```

`break` can be used to breaking out of a loop

Functions

```
def add(arg0, arg1):
    a = arg0 + arg1
    return a
```

- The keyword `def` indicates the start of a function
- Function arguments are listed separated by commas (by assignment)
- A colon (:) terminates the function definition
- Indentation is used to indicate the contents of the function (not optional)
- return is optional. If omitted, it takes the special value `None`

Classes

```
class stack():
  def __init__(self):
    self.items = []

  def push(self, x):
    self.items.append(x)

  def pop(self):
    x = self.items[-1]
    del self.items[-1]
    return x

  def empty(self):
    return len(self.items) == 0
```

Usage:

```
t = stack()
print t.empty()
t.push("hello")
print t.empty()
t.pop()
print t.empty()
```

Modules

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# today.py
import datetime
today = datetime.date.today()
print today
```

Run from a terminal

```
python today.py
```

```
today.py
```

And for the rest...

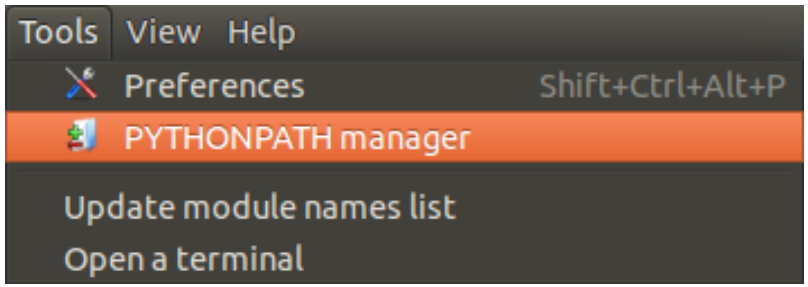
- Python has a very good official documentation.
- For each language-specific problem there is almost always a single “pythonic” community-approved solution
- Research and understand solutions (Stack Overflow)
- Encapsulate small problems, test in interpreter

Setting up PYTHONPATH

PYTHONPATH is an environment variable (or set of registry entries on Windows) that lists the directories Python searches for modules.
UNIX – .bashrc

```
export PYTHONPATH=${PYTHONPATH}:/path_to_library
```

If you use the IDE spyder you have to add the PYTHONPATH manually

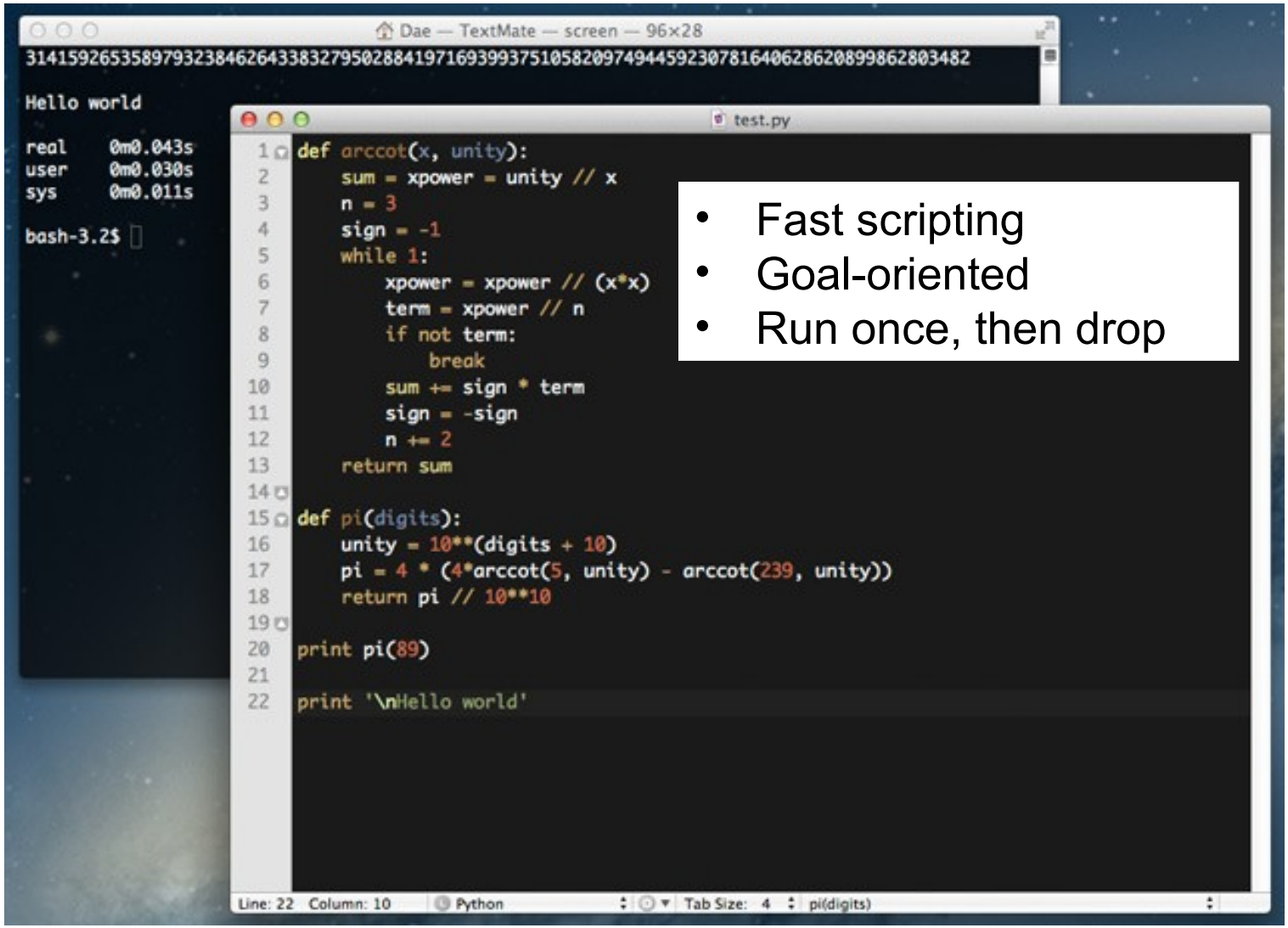


Resources

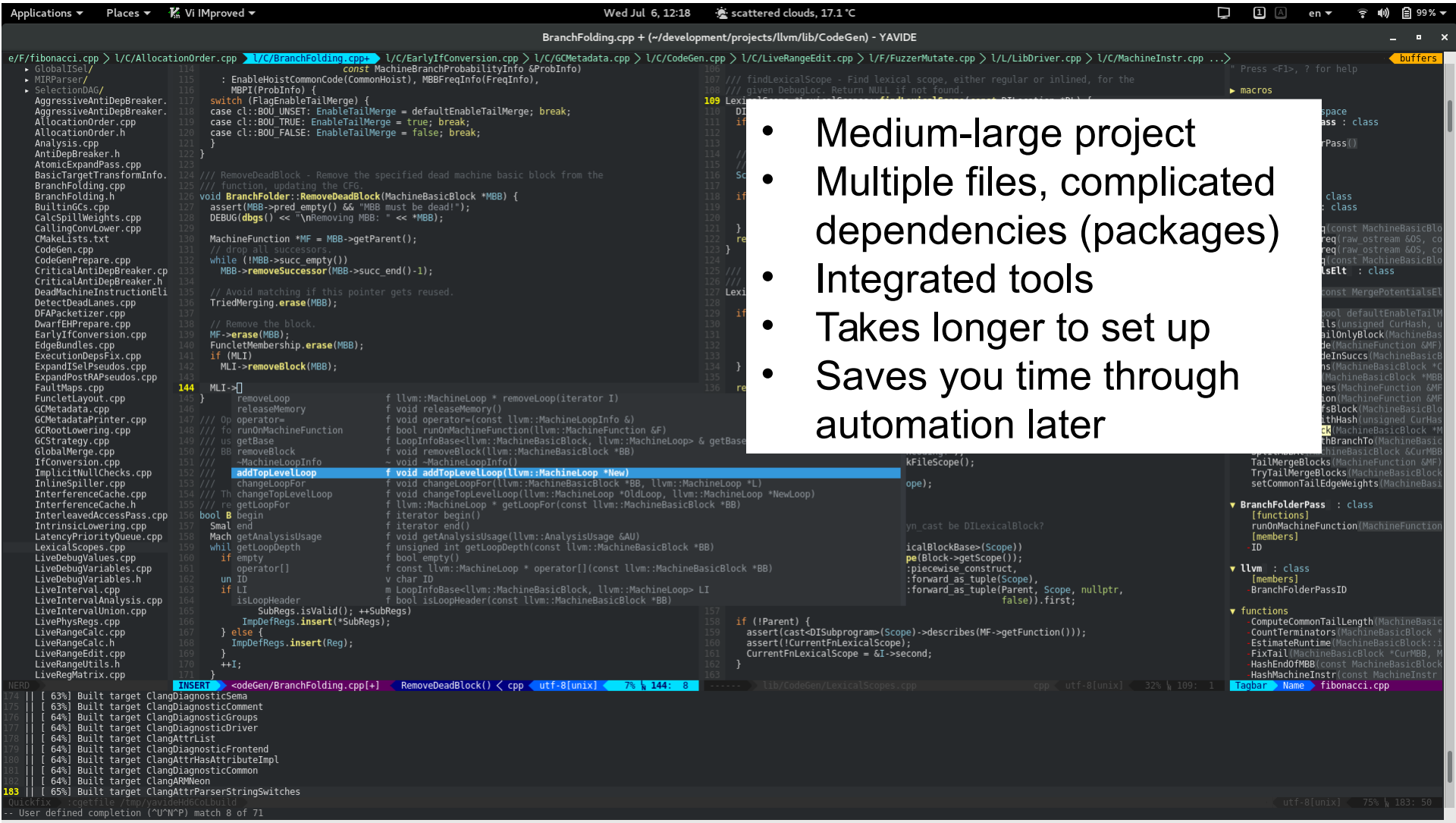
- https://www.python.org/
- https://www.python.org/about/gettingstarted/
- https://wiki.python.org/moin/BeginnersGuide/Overview
- https://pythonhosted.org/spyder/
- scipy https://www.scipy.org/
- numpy http://www.numpy.org/
- matplotlib http://matplotlib.org/
- numpy for MATLAB users https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

Ways to code in Python

Terminal + Editor



Integrated Development Environment (IDE)



- Medium-large project
- Multiple files, complicated dependencies (packages)
- Integrated tools
- Takes longer to set up
- Saves you time through automation later

Typical assists by an IDE

- Debugger
- Manage package dependencies
- Click on a class/package to “jump” to its declaration
- Autocomplete
- Syntax/Error Highlighting
- Refactoring
  - Mass-renaming
  - Moving files + references
  - Code reformatting (“tabs or spaces?”)
- Automated testing
- Git, Virtualenv, and other 3rd party tools and plugins

“So what do I need?”

- Answer: It depends on you and your project.
- Situation in research (from my experience):
  - More goal-oriented than in software-development
  - Methodology & results matter more than software quality
  - But small experiments can grow in unexpected ways
  - “I should have done this properly from the beginning”
- Suggestion: Find a middle ground!
  - Use an IDE for any multi-file project that requires more than one sitting to be completed
  - Use terminal/interpreter for “throwaway code”
  - ***Prioritize being time-efficient***

VS

+ More features
+ Better Customization
– slow to load

+ Simpler
– Limited
- Included in *Anaconda Python Distribution* (Linux/Windows)
- Similar to Matlab (Object & Variable explorer)