ACM/CS 114
Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

Spring 2012

Languages and programming paradigms

- a very active area of research
  - dozens of languages and runtime environments of the last 50 years
- the survivors:
  - procedural programming, and its offspring structured programming
  - functional programming
  - object oriented programming
- current areas of research:
  - component oriented programming
  - aspect programming
- languages are important:
  - they reflect an approach to computing
  - they shape what is easily expressible
- we'll take a quick tour of python
  - resources: `www.python.org`
  - overview of the language
  - interactive sessions with the interpreter
  - building extensions in `C/C++`

A python script

- python reads like pseudocode
- here is the code for the π estimator using Monte Carlo integration over the quarter disk

```
# get access to the random munber generator functions
import random
# sample size
N = 10**5
# initialize the interior point counter
interior = 0
# integrate by sampling some number of times
for i in range(N):
    # build a random point
    x = random.random()
    y = random.random()
    # check whether it is inside the unit quarter circle
    if (x*x + y*y) <= 1.0: # no need to waste time computing the sqrt
        # update the interior point counter
        interior += 1
# print the result:
print("pi: {0:.8f}".format(4*interior/N))
```

Overview

- built-in objects and their operators
  - numbers, strings, containers
  - files
- statements
  - evaluating expressions, explicit and implicit assignments, logic, iteration
- functions
  - scope rules, argument passing, callable objects
- modules and packages
  - name qualification, importing symbols
- user defined objects
  - declarations and definitions, inheritance, overloading operators
- exceptions
  - raising and catching, exception hierarchies

Syntax

- comments: from a `#` to the end of the line
- indentation denotes scope
  - avoid using tab characters; set your editor to insert a fixed number of spaces when the tab key is pressed
- statements end at the end of the line, or at `;`
  - open delimiters imply continuation
  - explicit continuation with \, but considered obsolete
- identifiers
  - start with an underscore or letter, followed by underscores, letters or digits
  - unicode is supported in identifier names; details at `http://docs.python.org/py3k/reference/lexical_analysis.html#literals`
  - identifiers are case sensitive
- certain classes of identifiers have special meaning
  - the pattern `__*__` is reserved by python for its own use
  - identifiers of the form `__*` in class definition are mangled and become private
  - identifiers of the form `_*` are not bulk imported from modules; more on this later

Reserved words

- the following words are reserved

```
False    None    True    and       as
assert   break   class   continue  def
del      elif    else    except    finally
for      from    global  if        import
in       is      lambda  nonlocal  not
or       pass    raise   return    try
while    with    yield
```

Built-in objects

- the more commonly used types

Type	*Sample*
booleans	`True`, `False`
numbers	`1234`, `3.14159`, `3+4j`
strings	`'help'`, `"hello"`, `"it's mine"`, `"""multi-line strings"""`
tuples	`(1, 'this', "other")`
lists	`['this', ['and', 0], 2]`
sets	`{1,2,3}`
dictionaries	`{'first': 'Jim', 'last': 'Brown'}`

- there are others; details to follow, as necessary

Operators and precedence

- from lower to higher precendece

Operator	*Description*
`lambda`	used to build anonymous functions
`if - else`	conditional expression (similar to `?:` from `C`)
`or`	boolean or
`and`	boolean and
`not`	boolean not
`in`, `not in`, `is`, `is not` `<`, `<=`, `>`, `>=`, `!=`, `==`	membership tests, identity tests, comparisons
`|`	bitwise or
`^`	bitwise xor
`&`	bitwise and
`<<`, `>>`	left and right bit shifts
`+`, `-`	binary addition, binary subtraction
`*`, `/`, `//`, `%`	multiplication, division, integer division, modulo
`+`, `-`, `~`	positive, negative, bitwise not
`**`	exponentiation
`[]`, `[:]`, `()`, `.`	indexing, slicing, function call, attribute reference

Numbers

- numeric literals

Literal	*Description*
`1234`	arbitrary precision integers
`3.1415`, `6.023e23`	floats
`j`, `1+j`	complex numbers
`0b1001`	binary integers
`0o777`	octal integers
`0xdeadbeef`	hexadecimal integers

- expressions:
  - the usual arithmetic operators
  - bitwise operators similar to `C`
  - adjust precedence and association by using parenthesis; be aware of the "tuple conflict"
  - in expressions with mixed types, python converts towards the wider types