ACM/CS 114
Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

Spring 2012

Comparisons

- the following are considered false in logical expressions
  - the boolean constant `False`
  - the special object `None`
  - the number `0`
  - any empty container
- other values are true, including the boolean constant `True`
- operators:
  - the object identity operator `is`
  - the container membership operator `in`
  - the usual relational operators – borrowed from `C`
- object comparisons
  - strings are compared lexicographically
  - nested data structured are checked recursively
  - lists and tuples are compared depth first, left to right
  - dictionaries are compared as sorted (key, value) tuples
  - user defined types can supply custom comparison functions by overloading the special methods

Assignments

- explicitly, using the = operator

```
greeting = 'Hello world!'
```

  which makes the symbol `greeting` become a name for the literal string in the right hand side

- implicitly, when defining a function

```
def greeting(name): pass
```

  which makes `greeting` become a name for the function object built out of the statements that follow the `:`

- implicitly, when defining a class

```
class greeting: pass
```

  which makes `greeting` become a name for the *class object* built out of the statements that follow the `:`

- implicitly, when importing symbols from a module

```
import sys
from math import pi
from math import pi as π
```

Selections

- using `if`

```
if <expression>:
    <statements>
elif <expression>:
    <statements>
else:
    <statements>
```

- no `switch` statement
  - use an `if` cascade
  - better yet, think about achieving the same result using containers; it's typically more efficient and robust

Iteration

- the `while` loop

```
while <expression>:
   <statements>
else:
   <statements>
```

executes the statements in its body until its expression evaluates to `False`, at which point it executes the optional `else` clause

- the `for` loop

```
for <name> in <expression>:
   <statements>
else:
   <statements>
```

evaluates its expression once to get an iterator, binds `name` to each object provided by the iterator and executes its body; iteration stops when the iterator is exhausted, at which point the `else` clause is executed; if execution encounters a `break` statement in the body of the loop, iteration is terminated, and the `else` clause is skipped; if execution encounters a `continue` statement, it skips the remainder of the loop body and proceeds with the next item from the iterator