
The Pyth Language

Lecture 5

Prof. Hilfinger CS164 Lecture 5

1

Administrivia

- Project #1 now available on-line
- Please make sure you have registered your team (and also have electronically registered with us as well)

Prof. Hilfinger CS164 Lecture 5

2

Historical Background

- Pyth comes from Python, a popular "scripting language"
- Python comes from ABC, a simple and powerful language for teaching & prototyping

Prof. Hilfinger CS164 Lecture 5

3

Features of Pyth

- Type-safe language, with both dynamic and static typing
- Object-oriented features based on exemplars
- Convenient built-in types for sequences, strings, and mappings (dictionaries)
- Clean, indentation-based statement grouping

Prof. Hilfinger CS164 Lecture 5

4

Program structure

- Program is a sequence of statements
- Each statement is either
 - One or more **simple statements** on a line, separated by ;'s, ending in newline
 - A **compound statement**
 - A **type declaration** (new in Pyth) + newline
 - An **import statement** + newline

Prof. Hilfinger CS164 Lecture 5

5

Simple Statements I: Pass

- Pass does nothing:

```
def f (n):  
    pass    # Must be statement here
```

Prof. Hilfinger CS164 Lecture 5

6

Simple Statements II: Print

- To print values separated by spaces:

```
print "x,y =", 3, 4
```



```
=> x,y = 3 4
```
- To print values without newline at end:

```
print "x,y =", # Extra comma does it
```

```
print 3; print 4
```



```
=> x,y = 3
```

```
4
```

Prof. Hilfinger CS164 Lecture 5

7

Printing to a file

```
print >> sys.stderr, "You made an error"
```

- Prints to file `sys.stderr` (the standard error output)
- Otherwise like ordinary print.

Prof. Hilfinger CS164 Lecture 5

8

Simple Statements III: Assignment

- Simple cases like C++ or Java:

```
x = 3; A[i] = 2; q,r = y + 2; z += 1
```
- But we also have:

```
a, b = 1, 10 # a=1; b = 10
```

```
(a,b) = 1, 10 # Same thing
```

```
x, a[0], y = a3ElementList
```

```
a, (b, c), d = [ 1, (2,3), 4 ]
```

Prof. Hilfinger CS164 Lecture 5

9

Compound Statements I: if

- Simplest form looks familiar (fewer {}'s):

```
if 0 > x > 20: print "too big"; x = 20
```

```
elif x > 10: print "OK"
```

```
else: print "too small"
```
- But only list of simple statements possible after ":" with this form

Prof. Hilfinger CS164 Lecture 5

10

Indentation and suites

- For more complicated "thens" or "elses", use indentation:

```
if x > 0:
```

```
    y = f(x)
```

```
if y > 0:
```

```
    print "y is", y
```

```
else: # Matches first if
```

```
    print "x is negative"
```

Prof. Hilfinger CS164 Lecture 5

11

Indentation and suites II

- Instead of { ... }, Pyth (like Python) uses indentation.
- General form:

```
Line with indentation N:
```

```
    Statement with indentation N'>N
```

```
More lines indented > N
```

```
Line with indentation N
```
- Each more-indented line adds a left bracket
- Each less-indented line adds a right bracket for each unbalanced more-indented lines

Prof. Hilfinger CS164 Lecture 5

12

Indentation and suites III

- Tabs indent to multiple of 8 spaces
- Inconsistent indenting is an error:

```
if x < 0:
    print x
    print y    # Error
```

Prof. Hilfinger CS164 Lecture 5

13

Compound Statements II: While

- While is almost as in Java, modulo parentheses and suites:

```
while n > 0:
    s += A[n]
    n -= 1;
```

- `break` and `continue` as in Java (but no label)

Prof. Hilfinger CS164 Lecture 5

14

While with else

- A new twist: end-of-loop code
- Executes only if test terminates loop:

```
while i < N:
    if P(A[i]): break
    i += 1
else:
    print "Error: didn't find it."
```

Prof. Hilfinger CS164 Lecture 5

15

Compound Statements III: For

- For loop is like Java 5's "for (String S: L)"
- Works for any type with `__getitem__` operation, including built-in sequences:

```
someList = [2, 3, 5, 7, 11, 13, 17];
for p in someList:
    if x % p == 0: break
    else: print "Maybe", x, "is prime?"
```

Prof. Hilfinger CS164 Lecture 5

16

Fancier for statements

- The for statement performs assignment statements to control variables, so...

```
pairs = (("boy", "girl"), ("fish", "bike"))
for left, right in pairs:
    print left, "is to", right, "as"
=> boy is to girl as
    fish is to bike as
```

Prof. Hilfinger CS164 Lecture 5

17

Importing

- In Pyth (not Python), importing is just textual inclusion:

```
import foo
```

- Looks for file named "foo.py" in any directory in "search path" (see project 1).
- Importing same name twice has no effect the second time
- Only allowed at outer level of program.

Prof. Hilfinger CS164 Lecture 5

18

Definitions I: Constants

- The declaration
`def name = expression`
evaluates `expression` and makes `name` a constant with that value.
- (This is not like Python)

Prof. Hilfinger CS164 Lecture 5

19

Definitions II: Constant functions

- To create a new function (or method) value:
`def gcd (x, y):`
 `if x == y: return x`
 `elif x > y: return gcd (x % y, y)`
 `else: return gcd (y, x)`
- Functions always return value, but it is the value `None` by default.

Prof. Hilfinger CS164 Lecture 5

20

Foreign functions

- To define a Pyth function with a C function:
`def newdir (name): import "mkdir"`
- We'll make extensive use of this to implement all the built-in methods of Pyth.

Prof. Hilfinger CS164 Lecture 5

21

Local variables and scope I

- Local variable is defined by assigning to it:
`outer = 2 # outer defined everywhere`
`def f (q): # q defined in body of f`
 `x = 2 # x defined in body of f`
 `def g ():`
 `x = 6 # NEW x, local to g`
 `print x, y # will print 6 3`
 `y = 3; g ()`
 `print outer, x # will print 2 2`

Prof. Hilfinger CS164 Lecture 5

22

Local variables and scope II: Global

- Can assign to `outer-level` variables in function by declaring them global:

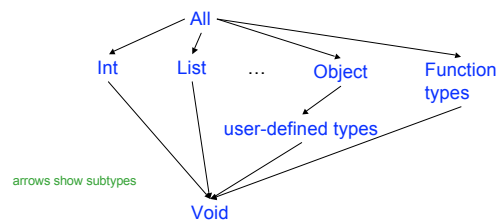
```
errs = 0 # process can change this
def process (x):
    global errs
    if x < 0: errs += 1; return
    ...
```

Prof. Hilfinger CS164 Lecture 5

23

Types and type declarations

- Pyth has a lattice of types:



Prof. Hilfinger CS164 Lecture 5

24

Types

- Types all have names:
 - Any
 - Int, Float, Bool, String, Tuple, Xrange, List, Dict, File, Object
 - Types introduced by user with "class..."
 - Function types: (Int, Int) -> Any
 - Void (the type of None)

Prof. Hilfinger CS164 Lecture 5

25

Dynamic and Static Types

- Every value has a type; types checked at runtime (at latest) for legal operations
- Every variable has a **static type**, constraining types of values it may contain (like Java, C, C++, etc.)
- The type of variable's value is its **dynamic type** (always a subtype of static type).
- All of this is just like Java

Prof. Hilfinger CS164 Lecture 5

26

Declaring Types

- By default, static type of variable, parameter, named constant is Any.
- def'd functions by default have type (Any,...,Any) -> Any
- Can declare static type of any of these with:
 - `x : Int`
 - `func : (Int, Int) -> Bool`
- Last one also gives parameters types

Prof. Hilfinger CS164 Lecture 5

27

Pre-Defined Types I: Simple Stuff

- Ints, Floats are as in Java
- Constant **None** is like null in Java
- Bool is like boolean in Java (constants **True**, **False**)
- String pretty much as in Java
 - But no "char" type: one-character strings double as characters

Prof. Hilfinger CS164 Lecture 5

28

Pre-Defined Types II: Sequences

- Strings, Tuples, Lists, and Xranges are all sequence types.
- That is, one can write `x[i]` to get *i*th character; negative indices count from right. `x[-1]` is last item.
- + is concatenation
- Can slice sequences:
 - `x[1: 3]` contains `x[1]`, `x[2]`
 - `x[2:]` contains everything from 2 on.

Prof. Hilfinger CS164 Lecture 5

29

Tuples

- Tuples are *immutable*: can't modify elements
- Created with expression lists (in ())'s if needed):
 - (2, "a string", True, None, (1,2))
 - () # Empty
 - (2,) # One element

Prof. Hilfinger CS164 Lecture 5

30

Lists

- Lists are mutable sequences.
- Create with *list display*:

```
[ ] # Empty  
[ 1, 2, "a string" ]
```
- Change with assignments:

```
L = [ ]; L += [1]; L += [3] # Now L=[1,3]  
L[1] = 5; L[0:1] = [] # L now [5]  
L[1:] = [9, 11, 13] # L now [5, 9, 11, 13]
```

Prof. Hilfinger CS164 Lecture 5

31

Xranges

- Xranges are immutable sequences of Ints.
- Useful in for loops:

```
for i in xrange(0, N):  
    k += i
```

Prof. Hilfinger CS164 Lecture 5

32

Dicts

- A Dict is a mutable mapping (like Java Map).
- Convenient syntax:

```
defns = { 'apple' : 'fruit', 'car' : 'machine' }  
defns['cow'] = 'animal'  
if 'cow' in defns: print defns['cow']  
for key in defns:  
    print key, '->', defns[key]
```

Prof. Hilfinger CS164 Lecture 5

33

User-defined Classes

- Pyth supports only single inheritance, no interfaces.
- To declare a class:

```
class Thing (ParentType):  
    instanceVar = 3  
    def instanceMethod (self, dir): ...  
    class def staticMethod (): ...  
    def __init__(self,x): ... #Constructor
```

Prof. Hilfinger CS164 Lecture 5

34

Using A Class

- Syntax for creating a Thing:

```
Thing(3)
```

creates a Thing and calls constructor (`__init__`) with new Thing and 3.
- Access to instance variables, methods, and class methods as in Java:

```
x.instanceVar, x.instanceMethod('n'),  
Thing.staticMethod(), x.staticMethod()
```

Prof. Hilfinger CS164 Lecture 5

35

Instance Methods I

- The "this" parameter is explicit in Pyth (and called "self" by convention):

```
class Cls (Object):  
    var = 0  
    def Meth (self, x): self.var += x
```
- Usual method-calling syntax works by special dispensation:

```
x.Meth(3) ==> (x.Meth)(x, 3)
```

Prof. Hilfinger CS164 Lecture 5

36

Instance Methods II: Alternate Syntax

- If a name `f` is not otherwise defined, then `f(x,...)` is transformed into `(x.f)(x, ...)`
- This strange convention is peculiar to Pyth and due entirely to your instructor's irritation with object-oriented syntax.

Prof. Hilfinger CS164 Lecture 5

37

Initialization and Exemplars I

- The class definition

```
class Child (Parent):  
    var = 3  
    def f(self, x): ...
```

creates a special *exemplar instance* of `Child`.
- Can refer to `var` in exemplar as `Child.var`

Prof. Hilfinger CS164 Lecture 5

38

Initialization and Exemplars II

- When you create a new `Child`, its value of `var` is initialized from `Child.var`
- As a result,

```
x1 = Child ()  
Child.var = 42  
x2 = Child ()  
print x1.var, x2.var
```

prints 3 42.

Prof. Hilfinger CS164 Lecture 5

39

Operators

- Most Pyth expression operators are actually just shorthand for function calls.
- For example:

```
x + y is same as __add__(x,y)  
x[i] is same as __getitem__(x,i)
```
- As a result, you can define these operators on your own classes.

Prof. Hilfinger CS164 Lecture 5

40