THE UNIVERSITY OF ARIZONA®

Department of Computer Science
College of Science

CS520 Principles of Programming Languages
Dr. Christian Collberg

Final Exam

May 14th, 2008

Raquel Torres Peralta
Federico Miguel Cirett Galán

History of Python

Python was designed by Guido van Rossum in the late 80's and early 90's. The name 'Python' is based on the famous BBC TV series of the 70's *Monty Python Flying Circus*.[1] Guido is a fan of this long ousted show, and the code examples in the documentation and tutorials of Python have plenty of references to it.

Rossum worked in the mid 80's in the programming language ABC, which targets computer-illiterate students, with a syntax that is easy to write. However, it was designed to write trivial programs. It had no I/O interactivity and no extensibility.

By 1986, van Rossum was working in the Amoeba distributed operating system project at CWI (**C**entrum voor **W**iskunde en **I**nformatics), and by the end of the decade, he realized that the project needed a merger between ABC and a shell script to do some administrative tasks in the Amoeba OS, since neither sufficed alone. So, in the Christmas break of 1989, he wrote the first draft of Python in only two weeks, borrowing some ideas from ABC, Modula-2 and Modula-3, with the goal in mind of a general-purpose scripting language that would help him with the Amoeba tasks.[2]

It took him over a year to have a working kit and in 1991 he released Python 0.90 to the USENET with a General Public License (GPL).

Python

Python is a very high level scripting multi-paradigm programming language; it supports object-oriented programming, structured programming, and functional programming.[3] It was designed with simplicity in mind. To improve the experience of the programmers with it, Python has the philosophy that "*There should be one -and preferably only one- obvious way to do it*".[4] As Python is a scripting language, the source files are run without compiling. The interpreter converts source code into bytecode just before run-time.

It is also possible to "pre-compile" a source file of python, but it will run at the same speed of a non-compiled file, because the bytecode will be the same. The only difference will be the load time, as the precompiled bytecode will be shorter than the source file.

If invoked without arguments, the Python interpreter runs as an interactive shell. The other running mode of Python is as an underlying runtime library. It is invoked with the Python binary and one or more modules, and a Python file to run.

Multiplatform

There are Python engines for every major computing platform in the market, including Windows, Linux, Unix, MacOS, PalmOS, OS/2, Amiga, and more. There's no need to modify a single line of code or to precompile again a bytecode file in order to run it in a different platform than the original. There are some caveats: the version of the interpreter must be the same, and there must be no calls to OS dependent modules.

---

[1] About Python. 2008. http://www.python.org/about/ [Retrieved on April, 2008]

[2] Interview with Guido van Rossum (July 1998) http://www.amk.ca/python/writing/gvr-interview [Retrieved on April, 2008]

[3] Wikipedia. *Python programming language* . 2008. http://en.wikipedia.org/wiki/Python_(programming_language) [Retrieved on April, 2008]

[4] PETERS, Tim. *The Zen of Python*. 2004. http://www.python.org/dev/peps/pep-0020/ [Retrieved on May 2008]

Semantics

Python semantics are very simple: functions and procedures are defined with a 'def' statement, plus the function name, arguments in parenthesis and a colon (:) at the end.

Flow control statements inside must be space indented and terminated with a colon, with all its enclosed statements indented.

This makes Python code very readable, almost as clear as plain English, with the added benefit that it is easier for the programmer to get his job done, as there are fewer characters to type.

Types and Objects

In Python everything is an object. Types are not an exception. When a variable is defined, an instance of its type is created. The type of an object cannot be changed once it's created, and sometimes, depending on its type, its value also cannot be changed. When the value of an object cannot be modified, it is said that the object is *immutable*. This is the case of tuples, numbers and strings. [5]

Since variables work as pointers to actual typed objects, they can change their type multiple times during the execution of the program.[6]

```
>>>x=1        # An int object is created. X points to it.
>>>type(x)
<type 'int'>
>>>x='Python' # A string object is created and x now points to it.
>>>type(x)
<type 'str'>
```

Data types

The language offers a set of built-in types to make our life easier. Among them, there is a set of sequence types: list, dictionary and tuples (a string is a tuple of chars). Also, different types can be defined in modules, even in other language like C, to be used as Python types.

Some Data Types Available in Python [7]

Data type	Description and examples
Complex	real and imaginary parts written as 3 + 4j or 1.23 - 0.0073j
Char. string	ordered collection of characters, enclosed by pairs of ', or " characters
List	ordered collection of objects, like [1,22,[321,'cow'],'horse']
Dictionary	collection of associated key:data pairs like {'first':'alpha', 'last':'omega'}
Tuples	similar to lists, like ('hen','duck',('rabbit','hare'),'dog','cat')
File	disk files as in: file1 = open('data.01','r'); data = file1.read()

[5] ROSSUM, Guido van. *Python Reference Manual - Objects, values and types* http://docs.python.org/ref/objects.html [Retrieved on April, 2008]

[6] BRUNNER, Robert.. "*Discover Python, Part 1: Python's built-in numerical types*". http://www.ibm.com/developerworks/opensource/library/os-python1/

[7] DYER , Charles. *The Python Programming Language*. 2002. http://pathfinder.scar.utoronto.ca/~dyer/csca57/book_P/node16.html [Retrieved on April, 2008]

In Python, variables don't need to be defined before use. In fact, a variable doesn't have a type by itself. Instead, it's a reference to an object of a certain type.

Lists, dictionaries and class instances are mutable, while strings, numbers and tuples are immutable. Python stores the type of object within itself and at run time checks if any operation applied to an object is permitted according to its type. Python performs *dynamic type checking*.[8]

Dynamic typing

Python performs type-checking at runtime. *type()* returns the type of an object:

```
def sum(a,b):
   if type(a) == str and type(b) == str:
        return a.upper()+ b.upper()
```

Notice that functions are polymorphic, since they can take arguments of any type. [9] Even when this approach gives us some freedom and code reusability, in some cases, we may want to make sure a function will accept arguments or return values of a certain type only. Then, we can make use of the *type-check module* [10] to specify a condition that the function must comply with in order to be executed, whether for the types of its arguments or the value it must return:

```
>>> from typecheck import accepts
>>> from typecheck import returns
>>> @accepts(Number)       # The function must accept a number as argument
>>> @returns(Number)       # The function must return a number
... def intFunction(a): # If a is not a number, the function will not be executed
...         return a
```

This is how we can have the freedom and comfort of dynamic typing and control of custom type-checking.

Objects

In Python everything is an object: built-in types (as lists, tuples and strings), functions, classes and obviously, instances of classes are objects as well. [11]

Every object has an identity, a type and a value. The identity of an object is no more than the address in memory where it is allocated. The type cannot be changed once the object is created. The value can or cannot change, depending on the type of the object.

---

[8] Wikipedia - *Python Programming/Data types*. 2008. http://en.wikibooks.org/wiki/Python_Programming/Data_types [Retrieved on April, 2008]

[9] Polymorphism is present extensively in Python types. See: Wikipedia -*Polymorphism in object-oriented programming*. 2008. http://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming#Python

[10] WINTER, Collin. *Type-checking module for Python*. http://oakwinter.com/code/typecheck/ [Retrieved on April, 2008]

[11] CHATURVEDI, Shalabh. *Python Types and Objects. 2005.* http://www.cafepy.com/article/python_types_and_objects/python_types_and_objects.html [Retrieved on April, 2008]

We can define an empty class, with the statement *pass* and define its attributes once an instance has been created:

```
class Point:
   def __init__(self, x, y):
   self.x = x
   self.y = y
```

```
class Paralellpiped:
    pass
```

The pass statement does nothing but fulfill the requisite of a statement present where it has to be. The class is defined with no attributes or methods.

```
>>> cube=Paralellpiped()
>>> cube.length = 10
>>> cube.center = Point(20, 20)
#length and Point are now attributes of Paralellpiped
```

Types, classes and functions are first-class entities.[12] A class can be defined with multiple base classes, giving place to multiple inheritance.

Python 2.2 includes the first phase of "type/class unification". This is a set of changes towards the unification of built-in types and user-defined classes. Here, the concept of new-style classes is introduced and it refers to a new way to handle classes. Some differences in new-style classes are the method resolution order and the disappearance of the restriction of using built-in types (like lists and dictionaries) as base in a class definition.

The method resolution order in the classic model is set by the left-to-right depth-first rule. There is a problem with this approach: In a "diamond diagram" of subclassing relationships, the method resolution may not provide an accurate result.[13]

Classic	**New-style** (All new-style classes have *object* or a built-in type as base)

`class A:`
`  def save(self): ...`
`class B(A):`
`  ...`
`class C(A):`
`  def save(self): ...`
`class D(B, C):`
`  ...`
	`class A(object):`
	`  def save(self): ...`
	`class B(A):`
	`  ...`
	`class C(A):`
	`  def save(self): ...`
	`class D(B, C):`
	`  ...`

In this case, the search order will be D, B, A, C, A. Here, D.save() will invoke A.save() instead of C.save().	In the new-style classes, the order changes. Now, the sequence is D, B, C, A, which results in the invocation of the C.save() method.

[12] SAYFAN, Gigi.. *Dig Deep into Python Internals. 2008.* http://www.devx.com/opensource/Article/31482 [Retrieved on May, 2008]

[13] ROSSUM, Guido van. *Unifying types and classes in Python 2.2.* 2008. http://www.python.org/download/releases/2.2.3/descrintro/#mro [Retrieved on May, 2008]

For now, the classic and new-style classes coexist in the 2.2 version.

Note: Objects are a wide subject in Python. There are a lot of features in this language on this topic, but unfortunately it is not possible to cover all of them in this writing.

Dictionaries, Lists and Tuples

A dictionary works like a map, where every value has a unique key (like a hash in Perl and like a Hashtable class in Java). Even when dictionaries are mutable, keys within them are not. [14]

```
>>> dct = {123:"First", 456:"second"}
>>> dct
{123: 'First', 456: 'second'}
>>> dct[123] # We can't access the keys by value, only by key.
'First'
```

These built-in types work like containers of a sequence of objects. Lists and tuples are very similar, but lists are mutable while tuples are not.

List:	Tuple:

`>>> l = [4,"home",2.0, "five", [5, 6]]`
`>>> l[2]`
`2.0`
`>>>l[1]=5`
	`>>> t = (4,"home",2.0, "five", (5, 6))`
	`>>> t[2]`
	`2.0`
	`>>>t[1]=5` #raise an error, since tuples are immutable

Tuples can contain variables and also can assign its values to a set of variables. These processes are known as *packing* and *unpacking:* [15]

Pack	Unpack

`>>> x = 1`
`>>> y = "some value"`
`>>> z = 44.0`
`>>> t = (x, y, z) # Pack the variables into a tuple`
`>>> t`
`(1, 'some value', 44.0)`
	`>>> x1, y1, z1 = t #Unpack the tuple into the named vars.`
	`>>> x1`
	`1`

Since tuples are not mutable, they are much faster than lists or dictionaries. This is the main reason for their inflexible nature.

Note: There are more built-in types not included here.

---

[14] PILGRIM, Mark. *Dive Into Python*. 2003. http://www.diveintopython.org/getting_to_know_python/dictionaries.html [Retrieved on April, 2008]

[15] BRUNNER, Robert. *Discover Python, Part 2: Explore the Python type hierarchy*. 2005. http://www.ibm.com/developerworks/opensource/library/os-python2/ [Retrieved on April, 2008]

Iterators

Iteration in Python is simple and easy to implement. Each time an iteration is needed, an Iterator object is created calling *Iter()*. Let's take look at the next code:

```
for element in (1, 2, 3):
    print element
```

This is a classic example of the printing of all the elements of a set. In this cycle, the *for* statement defines a set of three elements (*1,2,3*). It also invokes *iter()* implicitly. The returned iterator object will be referenced by element.

Each time the element is printed on line 2, the print statement invokes the next() method of the iterator object. This method generates the next value to make it available every time it is needed.

We can try it directly at the command line:

```
>>> l = 'abc'
>>> it = iter(l)  # Returns an iterator object
>>> it.next()
 'a'
>>> it.next()
 'b'
```

Generators

Generators are common functions that create and return iterators. A generator function contains the *yield* instruction (as in Ruby), that specifies the point of return of a value.

In the caller, when the next() method is invoked, the code in the generator function is executed from the last yield executed instruction (or the beginning of the function the first time), until a yield instruction or the end of the function is found.

This approach permits to hold and maintain the state of the function ready to be resumed the next time the *next()* method is called, improving performance.

```
def fib():
  a, b = 0, 1
  while 1:
    yield b     # Holds the state of current function
                # and returns an intermediate value
    a, b = b, a+b


def func():
  c=0
  b=fib()
  while c<10:
        print b.next()
        c=c+1
```

Scope rules

At every moment there are at least three namespaces in Python: Local (function level), Global (module level) and Python built-in namespace (language definition level). Names are resolved in the innermost enclosing function scope. [16]

Python is a statically scoped language, whose scope rules are similar to Algol's. The language supports nested scopes, where a variable defined in a function can be accessed by the functions defined within it.

Example:

```
>>> def Add1(x):
...     def Adder():
...             return x + 1
...     return Adder()
...
>>> Add1(5)
6
```

Note: Since this language does not require a declaration before use, the *global* statement must precede every global var name. Otherwise, a local variable will be created instead of modifying the existing global.

```
global.x = 10
```

Garbage collection

Python automatically performs garbage collection using the reference count algorithm. The flaw in this algorithm is reference cycling, which is not a problem for Python, as it can detect, most of the time, cyclic references and free objects when they are not useful, even when its reference counter is not equal to zero. [17]

```
>>> list1 = []
>>> list1.append(list1)
>>> del list1
```

In this example, a list is added to itself. In this case, list1 will be freed, even when its reference counter is 1.[18]

Garbage collection can be disabled with *gc.disable()* from the *gc* module, which also provides the easiness to configure the frequency of collection, have access to unreachable objects that cannot be freed and set options for debugging mode. [19]

---

[16] HYLTON, Jeremy**.** *Statically Nested Scopes*. 2007. http://www.python.org/dev/peps/pep-0227/ [Retrieved on May, 2008]

[17] There is no guaranty for 100% of success on cyclic references. See: http://docs.python.org/ref/objects.html

[18] *Python Documentation - Supporting cyclic garbage collection*. 2004. http://python.active-venture.com/ext/node24.html [Retrieved on May, 2008]

[19] *Python Library Reference*. 2008. http://docs.python.org/lib/module-gc.html [Retrieved on May, 2008]

Modules

Van Rossum had some experience programming in Modula-2 and Modula-3, and liked their easy extensibility, so he chose to make extensibility one of the major points of Python.[20]

He came up with the idea of dividing Python in modules, each module having its own namespace, variables, objects, procedures and functions. Each module would be contained in a script file.

When two modules are imported into the same project, they run side by side without affecting each other, because each module is self contained. At any point of the program, the reference to a function contained in a module must be preceded by its module name: *module_name.function_name()*, or assign it to a variable:

```
>>> import fibo      # import the fibo.py module
>>> fib = fibo.fib   # assign the function fib of the fibo module
                     # to fib local variable
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Python comes with a collection of built-in modules, a small set that is loaded by default at initialization time of the interpreter, and a big, really big library of modules for various tasks: networking, database access, data compression, email, XML, http and others, around 270 freely available in the latest installation of Python (2.5.2).

The official Python tutorial at python.org boast that the philosophy of this programming language is “batteries included”, meaning that everything you need to work (or have fun) with it is included.

Exception handling

Exception handling is based on Modula-3. In Python, for each thread, there is an error indicator of the last error encountered. When a function raises an error, it modifies this indicator.

Example[21]:

```
value = raw_input("Type a divisor: ")
try:
   value = int(value)
   print "42 / %d = %d" % (value, 42/value)
except ValueError:
     print "I can't convert the value to an integer"
except ZeroDivisionError:
     print "Your value should not be zero"
except:
     print "Something unexpected happened"
finally: print "Program completed successfully"
```

If no except block is matched, the error or exception is propagated back to the upper level, until it finds a matching except statement, or the top level Python interpreter that stops the

---

[20] VENNERS, Bill. The Making of Python - *A Conversation with Guido van Rossum, Part I* http://www.artima.com/intv/pythonP.html [Retrieved on April, 2008]

[21] Example from: http://www.freenetpages.co.uk/hp/alan.gauld/tutor2/tuterrors.htm [Retrieved on May, 2008]

program execution displaying a Python error message. [22]

The *finally* statement ensures that the code bellow it will always be executed, whether an exception is raised before or not. This may be useful when there are some open files and an exception rises. All files can be closed before execution is stopped.

Errors or exceptions can be generated manually, using the *raise* command. The raised error will look as any other exception to the program, and can be caught by a *try/except* block. [23]

```
denominator = input("What value will I divide 42 by?")

  if denominator == 0:
     raise ZeroDivisionError()
```

Python has predefined exceptions, but we can define our own exception types, defining a new exception class. In general, raising errors and managing exceptions in Python gives us a lot of freedom in control program flow.

Productivity

Python programs have great advantages: They are very easy to maintain, and require a lot less typing. In an interview with the magazine Artima developer, Guido van Rossum says that a program in Python has 5 times less code than one written in Java. [24]

Managing lists is where Python shines. Lists could be used as base for a stack or queue, and sorting lists is pretty simple. Even testing for an element in a list or tuple is trivial.

Examples of list uses:

```
>>> sports = [ 'Football', 'Baseball', 'Cricket']
>>> sports.sort()
>>> print sports
['Baseball', 'Cricket', 'Football']
>>> 'Cricket' in sports
True
```

Lists could be made up of any object valid in Python, and mixing integers, floats, strings, lists, tuples, dictionaries is common.

---

[22] GAULD, Alan. *Learning to Program* - Handling Errors. http://www.freenetpages.co.uk/hp/alan.gauld/tutor2/tuterrors.htm [Retrieved on May, 2008]

[23] Example from: http://www.freenetpages.co.uk/hp/alan.gauld/tutor2/tuterrors.htm [Retrieved on May, 2008]

[24] VENNERS, Bill. Programming at Python Speed- *A Conversation with Guido van Rossum, Part III* http://www.artima.com/intv/speed.html [Retrieved on April, 2008]

Functional programming and list comprehension

There's even space for functional programming in Python: map, filter and fold are included in the core of Python. But the preferred method for handling lists, the *pythonic way* is using list comprehension, as it is faster and easier to read:

```
>>> list = [ 1,2,3 ]
>>> list.append(42.5)
>>> list.pop()           #using the list as a Stack
42.5
>>> list.pop(0)          #using the list as a Qeue
1

>>> def square(x):
        return x*x
>>> map(square,list)     #map function to generate a squares list
4, 6
>>> [x*x for x in list]  #Easily matched with list comprehension
```

As a cool note, Paul Graham, in his online essay Great Hackers, wrote that the greater hackers program in Python, and disparages Java, stating that Python gives more freedom and productivity to hackers, and then, cites one hacker that built its own Segway riding machine and programmed its controller in Python in just one day. [25]

Performance

Performance is the first flaw of this language. For some tasks it can be tremendously slow.[26] Python is not recommended for heavily compute-bound applications.

Summary

Despite its slow performance, the language can be used for: portable applications, as "glue" between applications in different languages and platforms and for building working prototypes.

Python is a very easy-to-use, highly productive scripting language, being widely used. In fact, Google, Industrial Light & Magic, Yahoo, Zope, NASA and other heavy weights use Python in production due to its multiplatform capability, its ease of use and low maintenance costs.

---

[25] GRAHAM, Paul. Great Hackers. 2004. http://www.paulgraham.com/gh.html [Retrieved on April, 2008]

[26] NORVING, Peter. *Python for Lisp Programmers - Introducing Python*. http://norvig.com/python-lisp.html [Retrieved on April, 2008]

Table of Contents

Other references:

Wikipedia - *ABC programming language*. 2008.
http://en.wikipedia.org/wiki/ABC_programming_language [Retrieved on April, 2008]

VENNERS, Bill. *The Making of Python: A Conversation with Guido van Rossum, Part I*. 2003.
http://www.artima.com/intv/pythonP.html [Retrieved on April, 2008]

VENNERS, Bill. *The Making of Python: A Conversation with Guido van Rossum, Part II*. 2003.
http://www.artima.com/intv/pyscale.html [Retrieved on April, 2008]

PEMBERTON, Steven. *A short introduction to ABC programming language*. 2006.
http://homepages.cwi.nl/~steven/abc/ [Retrieved on April, 2008]

ROSSUM, Guido van. *Python Reference Manual - Object, values and types*. 2008.
http://www.python.org/doc/current/ref/objects.html [Retrieved on April, 2008]