Introduction to Programming using PYTHON

Session 2

N. D. Mendes
ndm@algos.inesc-id.pt

INESC-ID

October 9, 2006

Why Python?

- Easy to learn
- A growing community of developers
- An increasing set of tools made available to the bioinformatics community

Other languages

- PERL
- C
- C++
- Java
- ...

Python

- Interpreted language (sort of)
- Multi-paradigm (imperative, object-oriented, functional)
- Typed laguage, dynamic binding with some implicit coercions

Python

- Interpreter can be executed:
  - Interactively
  - From a script file

Python

- Data in Python is represented by Objects or relations amongst Objects

Python

- Data in Python is represented by Objects or relations amongst Objects
- Every Object has an identity, a type and a value

Python

- Data in Python is represented by Objects or relations amongst Objects
- Every Object has an identity, a type and a value

```
>>> a = 3
>>> id(a)
140930456
>>> type(a)
<type 'int'>
>>> a
3
>>>
```

Python

- Data in Python is represented by Objects or relations amongst Objects
- Every Object has an identity, a type and a value

```
>>> a = 3
>>> id(a)
140930456
>>> type(a)
<type 'int'>
>>> a
3
>>>
```

- Typed laguage, dynamic binding with some implicit coercions

Identifiers and Variables

Identifiers

An identifier in Python is a sequence of characters from the alphabeth $\{\_, a, \ldots, z, A, \ldots, Z, 0, \ldots, 9\}$ such that the first character of the sequence is in $\{\_, a, \ldots, z, A, \ldots, Z\}$.

Valid identifiers	**Invalid identifiers**
_	0
__id__	0beron
variable	1mpressive
f01234	elegant − and − readable

An identifier cannot be a keyword of the language
Identifiers are case-sensitive

Identifiers and Variables

List of keywords

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	yield	

Basic Types

- None
- Numbers
  - Integers
    - Plain integers (e.g. `10`)
    - Long integers (e.g. `1000000000L`)
    - Booleans (e.g. `True == 0`, `False == 1`)
  - Floating-point numbers (e.g. `3.1415`, `3.4e5`)
  - Complex numbers (e.g. `3+1j`)
- Sequences
  - Immutable sequences
    - Strings (e.g. `'this is a string\n'`)
    - Tuples (e.g. `(1, 3.0, [1,2], 'hello')`)
  - Mutable sequences
    - Lists (e.g. `[1, 3.0, [1,2], 'hello']`)
- Mappings
  - Dictionaries (e.g. `{'camel':'an interesting animal', 'ten':10, 'empty list':[]}`)
- Callable types
- ...

Operators and Precedence

`+x, -x, x ** y`
`x * y, x / y, x % y, x // y`
`x + y, x - y`
`x < y, x <= y, x > y, x >= y, x == y, x != y,` `x <> y, x is y, x is not y, x in s, x not in s`
`not x`
`x and y`
`x or y`

Operators and Precedence

`+x, -x, x ** y`
`x * y, x / y, x % y, x // y`
`x + y, x - y`
`x < y, x <= y, x > y, x >= y, x == y, x != y,` `x <> y, x is y, x is not y, x in s, x not in s`
`not x`
`x and y`
`x or y`

What is the result of evaluating the following expression:

```
0 and 1 or 2 + 4 * 5 and 5 or 2 + 3 or 5 - 3
```

Simple Statements

- Expression statements

e.g.

```
4 * int(user_function(a,b)) or 5
```

Simple Statements

- Expression statements
- Assignment statements

e.g.

```
a = 'a string'    b = 10        a,b = b,a

a,b = [0,1]       a,b = 'xy'    [a,b],c = [1,2],3

b += 5            b *= 2        a += ' another string'
```

Simple Statements

- Expression statements
- Assignment statements
- Pass statement

e.g.

```
pass
```

It does nothing

Simple Statements

- Expression statements
- Assignment statements
- Pass statement
- Del statement

e.g.

```
del a
del a,b,c
del s[0]
del s[2:5]
```

Eliminates the binding of the names passed as argument. or elements of sequences

Statements, Expressions and Functions

Simple Statements

- Expression statements
- Assignment statements
- Pass statement
- Del statement
- Print statement

e.g.

```
print "Hello, my name is", myname

print "This line has no newline character",

print >> file, "This prints to file, which is a file-like object",
```

Simple Statements

- Expression statements
- Assignment statements
- Pass statement
- Del statement
- Print statement
- Return statement

e.g.

```
return (x,y,z)
```

To be used only inside function blocks

Simple Statements

- Expression statements
- Assignment statements
- Pass statement
- Del statement
- Print statement
- Return statement
- Import statement

e.g.

```
import sys
import string as str
from string import *
```

Statements, Expressions and Functions

Simple Statements

- Expression statements
- Assignment statements
- Pass statement
- Del statement
- Print statement
- Return statement
- Import statement
- ...

We will see other statements in the next sessions

Expressions

Arithmetic Operations Coercion Rules

1. If either argument is Complex, the other is converted to Complex
2. Otherwise, if either argument is a Floating Point number, the other is converted to Floating Point
3. Otherwise, if either argument is a Long Integer, the other is converted to Long Integer
4. Otherwise both arguments are Plain Integers

Expressions

Atoms

- Identifiers
- Literals (e.g. `10`, `'string'`, `[]`)
- Enclosures
  - Parenthesized forms (eg. `()`, `(4+5)`, `(2+2,'a string')`)
  - List Displays (e.g. `[x for x in range(100)]`, `[x*y for x in range(10) for y in range(10) if x+y < 50]`)
  - Dictionary Displays (e.g. `{key1:value1, key2:value2}`)
  - String Conversions (e.g. `` `4+5` ``)

Statements, Expressions and Functions

Expressions

Primaries

- Atoms
- Attribute References (e.g. `sys.argv`)
- Subscriptions (e.g. `s[2]`, `a[-1]`, `d['key']`)
- Slicings (e.g. `a[:]`, `a[2:]`, `a[4:-1]`, `a[1::2]`)
- Calls (wil be dealt with in later sessions)
- Primaries are combined using the operators already discussed

An encoded string

```
encoded = 'abcdcefagfhijkcjh'
```

An encoded string

```
encoded = 'abcdcefagfhijkcjh'
```

```
from string import *
```

An encoded string

```
encoded = 'abcdcefagfhijkcjh'
```

```
from string import *
```

```
decoded = replace(replace(encoded, 'a','O'),'b','r') ...
```

An encoded string

```
encoded = 'abcdcefagfhijkcjh'
```

```
from string import *


trans = maketrans('abcdefghijk','orign fspec')
decoded = translate(encoded, trans)
decoded = title(decoded)
```

For next session

- From the manual
  - Read chapters 4, 5 ,6, 7
- Exercises
  1. Generate the list of the first 1000 multiples of 3
  2. Generate a string with 1000 'a' characters
  3. Given a string with charactes 'a', 't', 'g' and 'c', obtain the complemented string
  4. Now obtain the reversed-complemented string
  5. Given a string, obtain a new string with the characters of the original string occupying even positions
  6. Given the string `'atttggcggttatgggggggaaaat'` obtain the positions of the first three occurrences of `'at'`