Essential Computing for Bioinformatics

Lecture 1

First Steps in Computing

Bienvenido Vélez
UPR Mayaguez

Reference: How to Think Like a Computer Scientist: Learning with Python Ch 1-2

Outline

- Course Description
- Educational Objectives
- Major Course Modules
- First steps in computing with Python

Course Description (Revised)

This course provides a broad introductory discussion of essential computer science concepts that have wide applicability in the natural sciences. Particular emphasis will be placed on applications to BioiInformatics. The concepts will be motivated by practical problems arising from the use of bioinformatic research tools such as genetic sequence databases. Concepts will be discussed in a weekly lecture and will be practiced via simple programming exercises using Python, an easy to learn and widely available scripting language.

Educational Objectives (Revised)

- Awareness of the mathematical models of computation and their fundamental limits
- Basic understanding of the inner workings of a computer system
- Ability to extract useful information from various bio-informatics data sources
- Ability to design computer programs in a modern high level language to analyze bio-informatics data.
- Ability to transfer information among relational databases, spreadsheets and other data analysis tools
- Experience with commonly used software development environments and operating systems

Major Course Modules

Module	Hours
First Steps in Computing	3
Using Bioinformatics Data Sources	6
Mathematical Computing Models	3
High-level Programming (Python)	12
Extracting Information from Database Files	6
Relational Databases and SQL	6
Other Data Analysis Tools	3
TOTAL	39

Important Topics will be Interleaved Throughout the Course

- Programming Language Transalation Methods
- The Software Development Cycle
- Fundamental Principles of Software Engineering
- Basic Data Structures for Bioinformatics
- Design and Analysis of Bioinformatics Algorithms

First Steps in Computing

- Need a mechanism for expressing computation
- Need to understand computing in order to understand the mechanism
- Solution: Write your first bioinformatics program in a very high level language such as:

python

www.python.org

Solves the Chicken and Egg Problem!

Main Advantages of Python

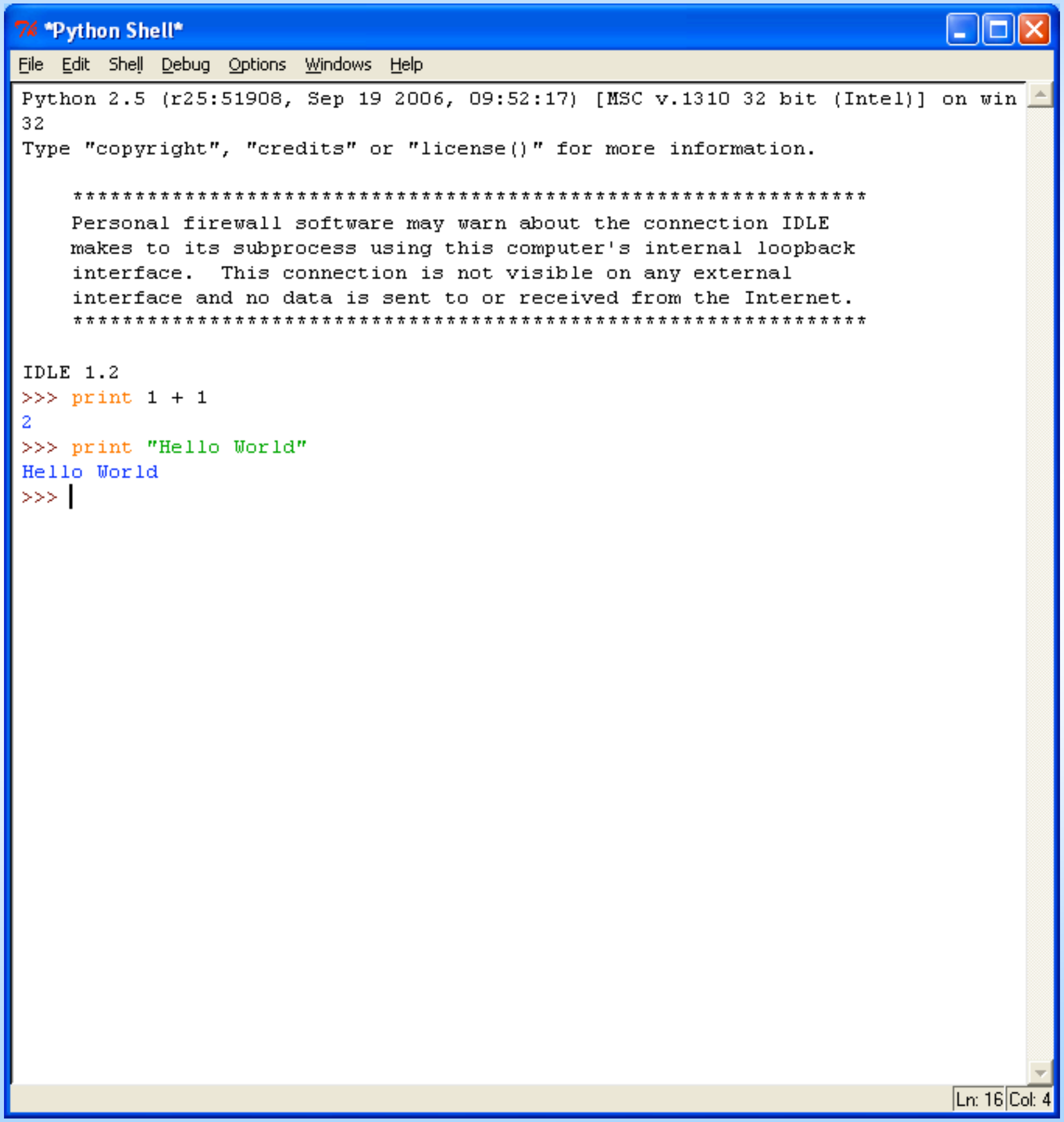
- Familiar to C/C++/C#/Java Programmers
- Very High Level
- Interpreted and Multi-platform
- Dynamic
- Object-Oriented
- Modular
- Strong string manipulation
- Lots of libraries available
- Runs everywhere
- Free and Open Source
- Track record in Bio-Informatics (BioPython)

Downloading and Installing Python on a Windows XP PC

- Go to www.python.org
- Go to DOWNLOAD section
- Click on Python 2.5 Windows installer
- Save ~10MB file into your hardrive
- Double click on file to install
- Follow instructions
- Start -> All Programs -> Python 2.5 -> Idle

Most Unix Systems today have Python pre-installed

Idle: The Python Shell



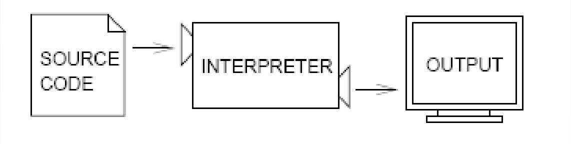
```
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2
>>> print 1 + 1
2
>>> print "Hello World"
Hello World
>>> 
```

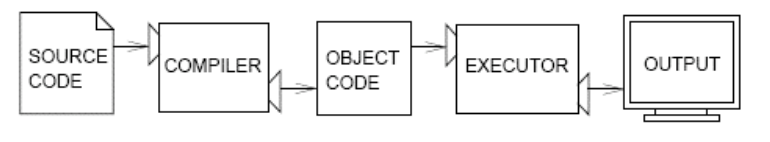
PL Translation Methods

Interpretation



- Run and Translate Simultaneously

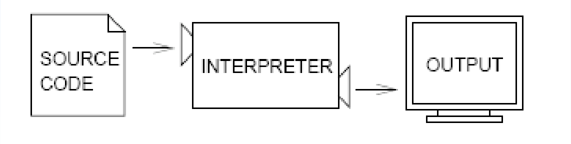
Compilation



- Translate to executable
- Then Run

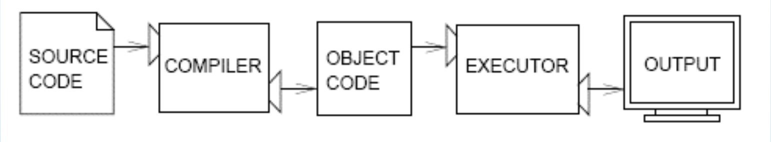
PL Translation Methods

Interpretation



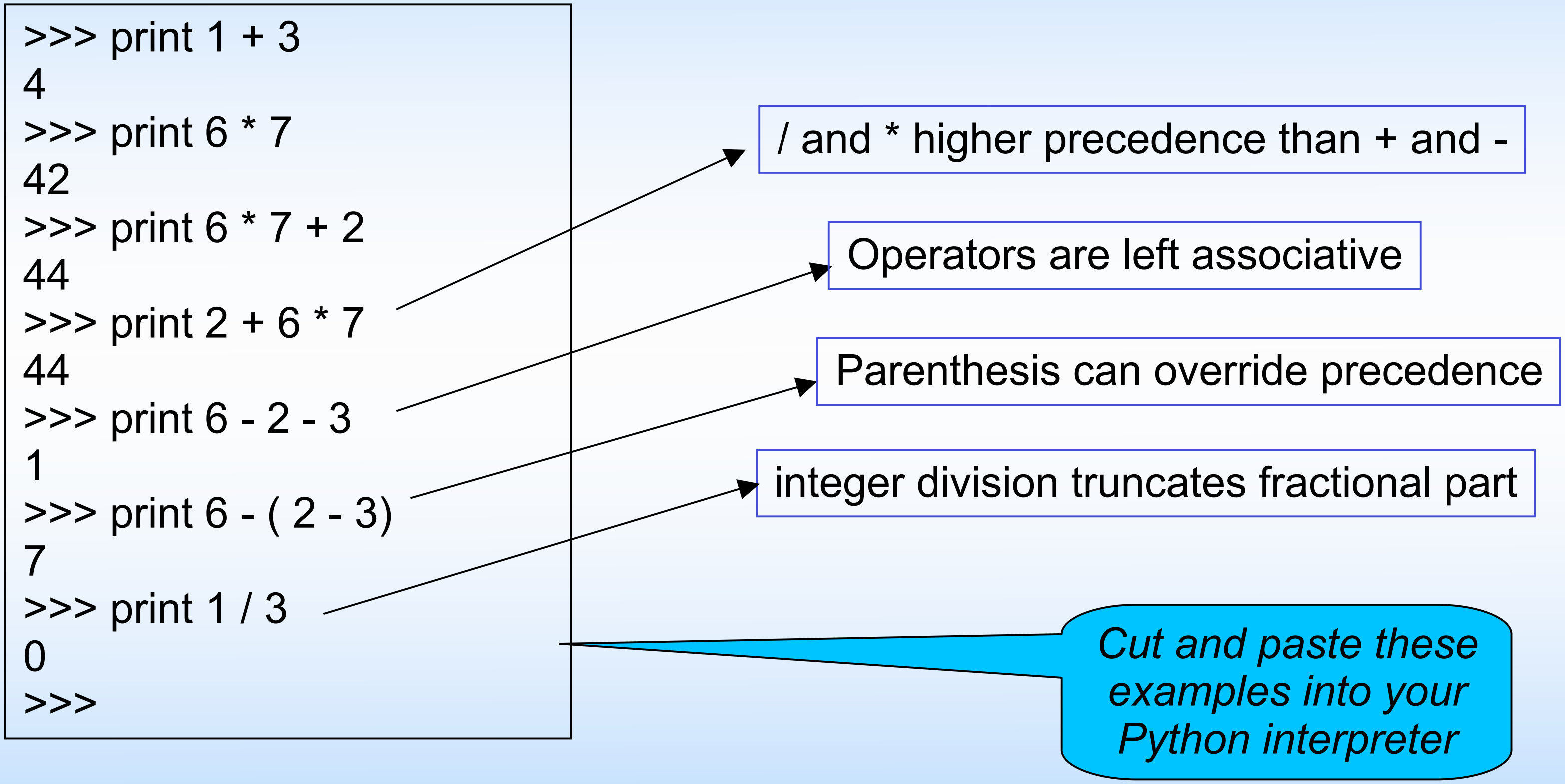
- Faster write-execute cycle
- Easier debugging
- Portable

Compilation



- Some errors caught before running
- Faster Execution

Python as a Number Cruncher
Integer Expressions



```
>>> print 1 + 3
4
>>> print 6 * 7
42
>>> print 6 * 7 + 2
44
>>> print 2 + 6 * 7
44
>>> print 6 - 2 - 3
1
>>> print 6 - ( 2 - 3)
7
>>> print 1 / 3
0
>>>
```

/ and * higher precedence than + and -

Operators are left associative

Parenthesis can override precedence

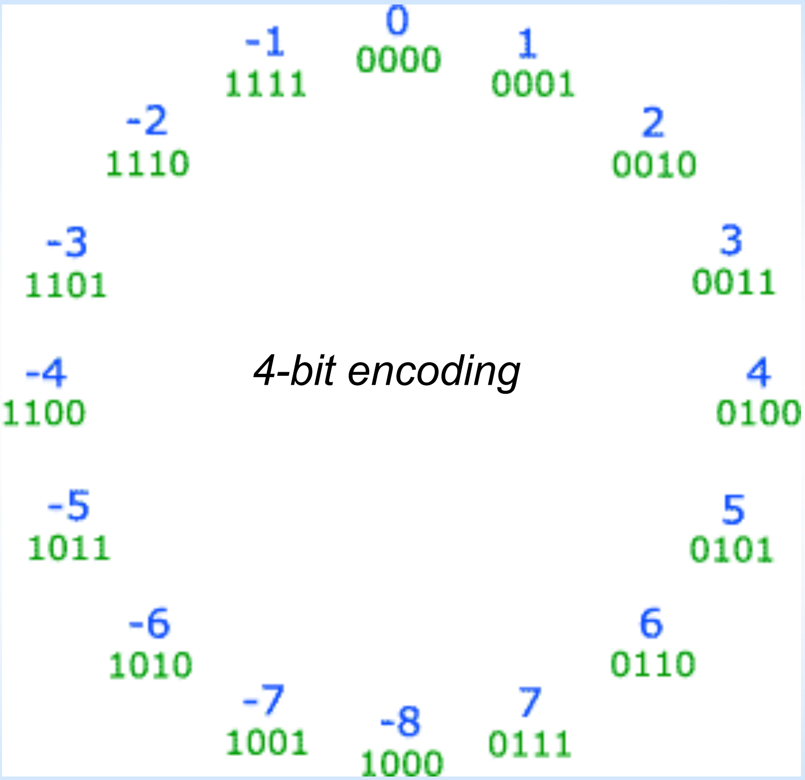
integer division truncates fractional part

Cut and paste these examples into your Python interpreter

Integer Numbers and Real Numbers
are DIFFERENT types of values

Integer Numbers

Two's Complement Encoding



0 0000

1 0001

2 0010

3 0011

4 0100

5 0101

6 0110

7 0111

-8 1000

-7 1001

-6 1010

-5 1011

-4 1100

-3 1101

-2 1110

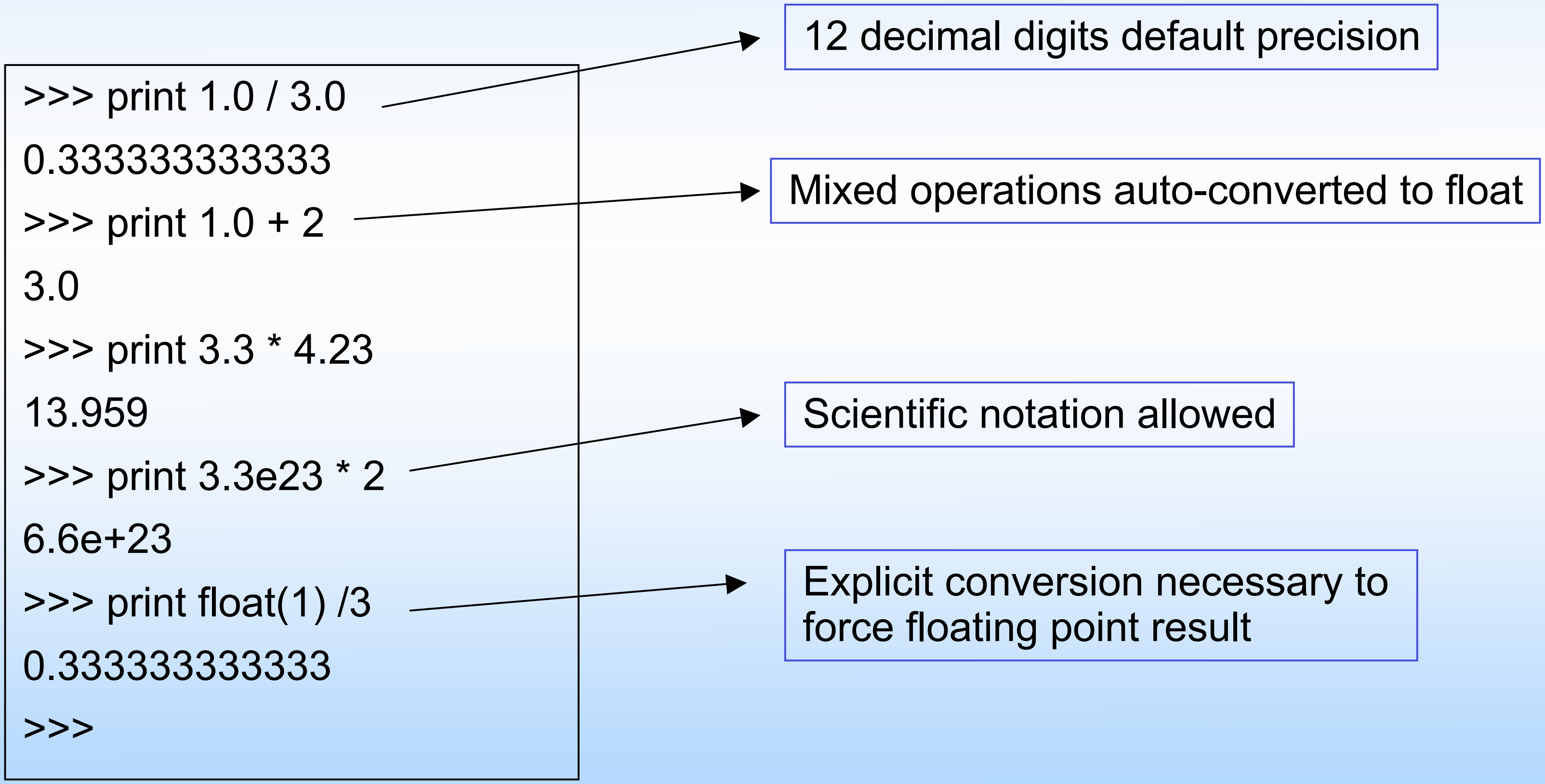
-1 1111

4-bit encoding

- *Half of the codes for positives and zero*
- *Half of the codes for negatives*
- *Negatives always start with 1*
- *Positives always start with 0*
- *Largest positive = $2^{(n-1)}$ -1, n = # of bits*
- *Smallest negative = $-2^{(n-1)}$, n = # of bits*
- *In binary addition ⇒ $2^{(n-1)} -1 + 1 = -2^{(n-1)}$,*

For Computer Engineering Convenience
All Data Inside the Computer is Encoded in Binary Form

Floating Point Expressions



```
>>> print 1.0 / 3.0
0.333333333333
>>> print 1.0 + 2
3.0
>>> print 3.3 * 4.23
13.959
>>> print 3.3e23 * 2
6.6e+23
>>> print float(1) /3
0.333333333333
>>>
```

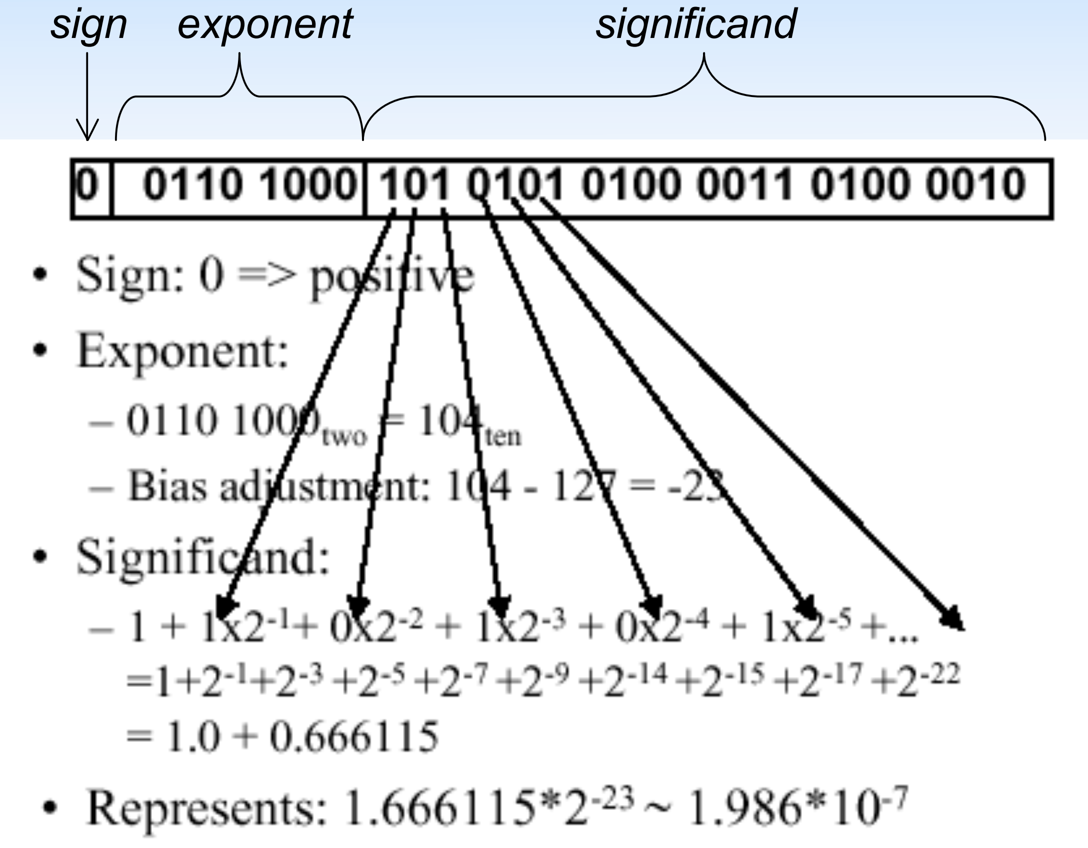
12 decimal digits default precision

Mixed operations auto-converted to float

Scientific notation allowed

Explicit conversion necessary to force floating point result

What is a Floating Point Value?



0 | 0110 1000 | 101 0101 0100 0011 0100 0010

- Sign: 0 => positive
- Exponent:
  - 0110 1000_{two} = 104_{ten}
  - Bias adjustment: 104 - 127 = -23
- Significand:
  - $1 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3} + 0x2^{-4} + 1x2^{-5} + ...$
    $=1+2^{-1}+2^{-3}+2^{-5}+2^{-7}+2^{-9}+2^{-14}+2^{-15}+2^{-17}+2^{-22}$
    = 1.0 + 0.666115
- Represents: $1.666115*2^{-23} \sim 1.986*10^{-7}$

- *Precision limited by number of bits in significand*
- *Range limited by number of bits in exponent*
- *Different behavior form base 10 floating point*
- *Some number that require many significand bits in base 10 may only require a few bits in base 2 to be represented exactly*
- *Rounding in base 2 may not yield intuitive results*

Virtually all systems use this IEEE 754 Floating Point Standard

String Expressions

```
>>> print "aaa"
aaa
>>> print "aaa" + "ccc"
aaaccc
>>> len("aaa")
3
>>> len ("aaa" + "ccc")
6
>>> print "aaa" * 4
aaaaaaaaaaaa
>>> "aaa"
'aaa'
>>> "c" in "atc"
True
>>> "g" in "atc"
False
>>> "act" [1]
'c'
```

+ operator concatenates string

len is a function that returns an integer representing the length of its argument string

any string expression can be an argument

* operator replicates strings

a value is an expression that yields itself

in operator finds a string inside another
And returns a boolean result

[ ]'s can be used to extract individual characters from strings

Strings are great for representing DNA!

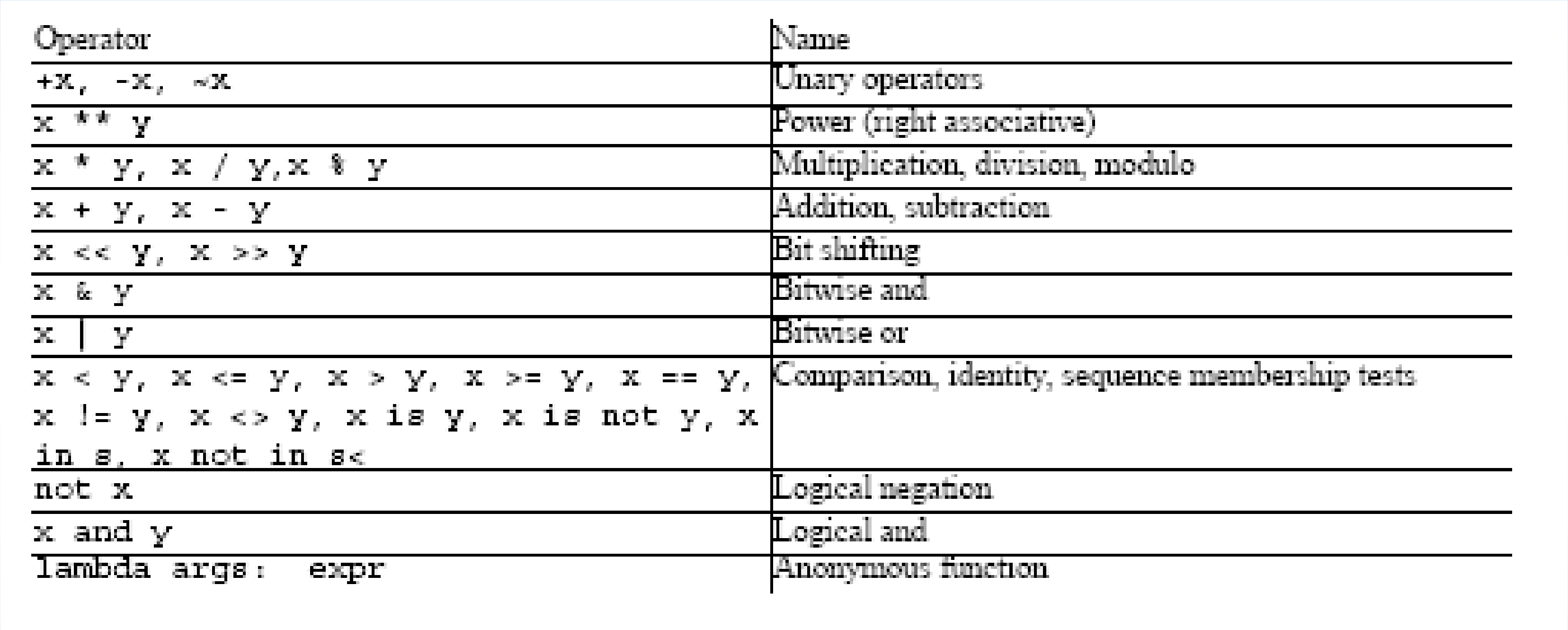
Preview of Functions

<function_name> (<arg1>, …, <argn>)

- Functions receive zero or more *arguments*
- Arguments are expressions that yield values
- Functions *return* a single object
- The function call is itself and expression that yields the object returned by the function
- The behavior of a function is established by an unwritten "contract"
- Example: The *len* function in Python receives one argument that must yield a string value. The function returns and integer value representing the number of characters in the string
- If the programmer violates the contract the function does not have to behave properly

We will spend lots of time talking about functions later in the course

Operator Precedence Rules



Operator	Name
`+x, -x, ~x`	Unary operators
`x ** y`	Power (right associative)
`x * y, x / y,x % y`	Multiplication, division, modulo
`x + y, x - y`	Addition, subtraction
`x << y, x >> y`	Bit shifting
`x & y`	Bitwise and
`x	y`	Bitwise or
`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<`	Comparison, identity, sequence membership tests
`not x`	Logical negation
`x and y`	Logical and
`lambda args: expr`	Anonymous function

What is the difference between and OPERATOR and a FUNCTION?

Table taken from Introduction to Programming Using Python

Statements vs. Expressions

- Expressions yield values
- Statements do not
- All expressions can be used as single statements
- Statements cannot be used in place of expressions
- When an expression is used as a statement, its value is computed yet ignored by the interpreter
- A "program" or "script" is s sequence of statements

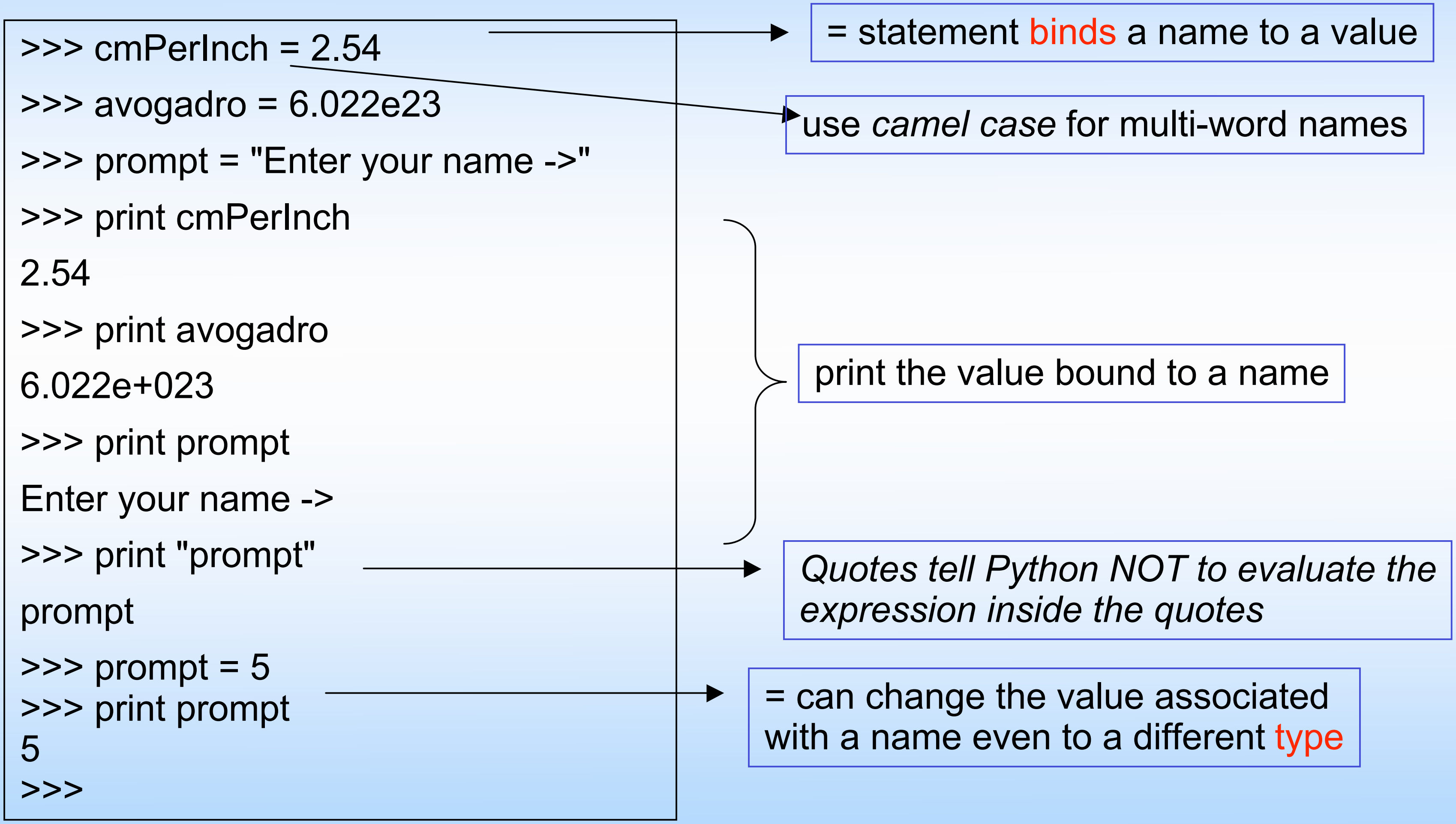
Expressions:

```
5
avogadro
"Hello"
len(seq)
```

Statements:

```
print "Hello"
avogadro=6.022e23
"Hello"
len(seq)
```

Values Can Have (MEANINGFUL) Names



```
>>> cmPerInch = 2.54
>>> avogadro = 6.022e23
>>> prompt = "Enter your name ->"
>>> print cmPerInch
2.54
>>> print avogadro
6.022e+023
>>> print prompt
Enter your name ->
>>> print "prompt"
prompt
>>> prompt = 5
>>> print prompt
5
>>>
```

= statement binds a name to a value

use *camel case* for multi-word names

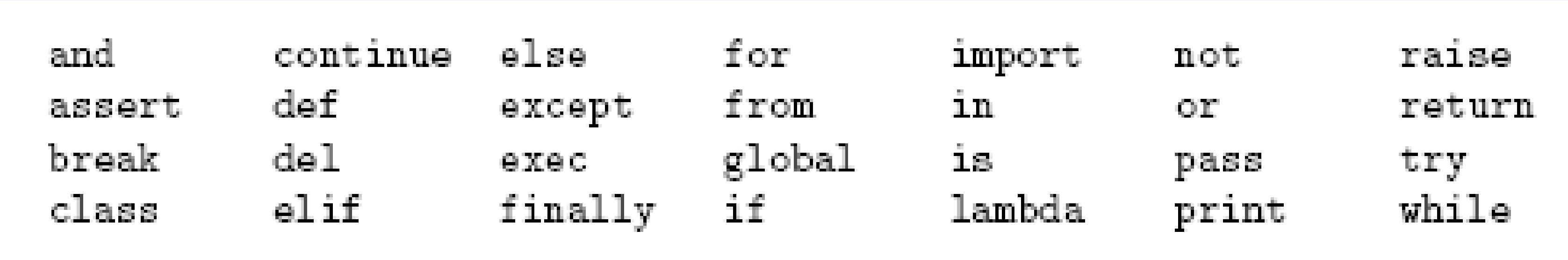
print the value bound to a name

Quotes tell Python NOT to evaluate the expression inside the quotes

= can change the value associated with a name even to a different type

Naming values is the most primitive abstraction mechanism provided by PL's

Python's 28 Keywords
Cannot be used as names



```
and       continue  else      for       import    not       raise
assert    def       except    from      in        or        return
break     del       exec      global    is        pass      try
class     elif      finally   if        lambda    print     while
```

Do not use these as names as they will confuse the interpreter

Values Have Types

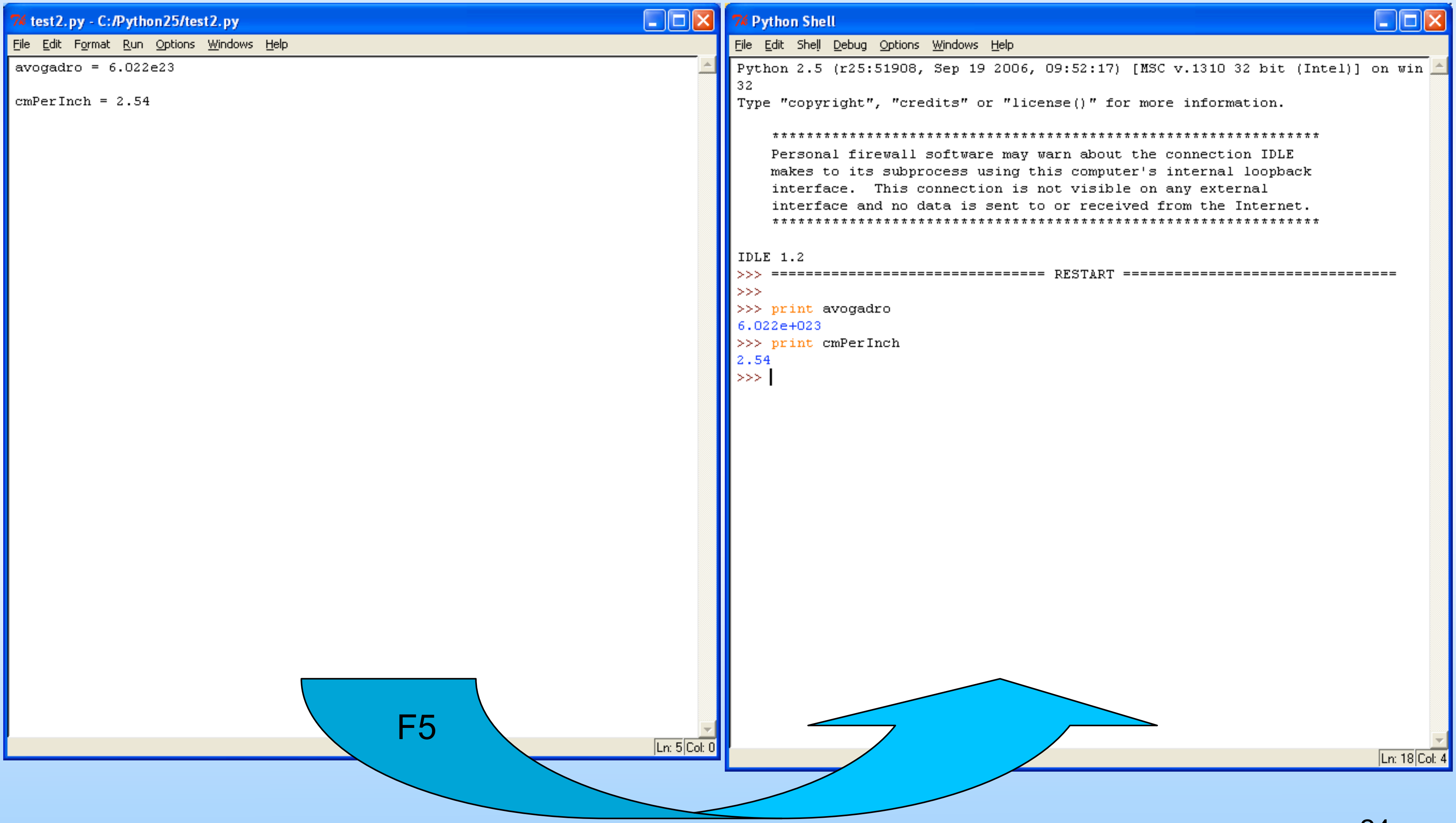
```
>>> type "hello"
SyntaxError: invalid syntax
>>> type("hello")
<type 'str'>
>>> type(3)
<type 'int'>
>>> type(3.0)
<type 'float'>
>>> type(avogadro)
<type 'float'>
>>> type (prompt)
<type 'int'>
>>> type(cmPerInch)
<type 'float'>
```

type is another function, not an operator

the "type" is itself a value

The type of a name is the type of the value bound to it

How Do I Run My Programs?


test2.py - C:/Python25/test2.py
File Edit Format Run Options Windows Help
Ln: 5 Col: 0
Python Shell
File Edit Shell Debug Options Windows Help
Ln: 18 Col: 4
F5

```
avogadro = 6.022e23

cmPerInch = 2.54
```

```
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2
>>> ================================ RESTART ================================
>>> 
>>> print avogadro
6.022e+023
>>> print cmPerInch
2.54
>>> 
```

Using Strings to Represent DNA Sequences

```
>>> codon="atg"
>>> codon * 3
'atgatgatg'
>>> seq1 ="agcgccttgaattcggcaccaggcaaatctcaaggagaagttccggggagaaggtgaaga"
>>> seq2 = "cggggagtggggagttgagtcgcaagatgagcgagcggatgtccactatgagcgataata"
>>> seq = seq1 + seq2
>>> seq
'agcgccttgaattcggcaccaggcaaatctcaaggagaagttccggggagaaggtgaagacggggagtgggg
agttgagtcgcaagatgagcgagcggatgtccactatgagcgataata'
>>> seq[1]
'g'
>>> seq[0]
'a'
>>> "a" in seq
True
>>> len(seq1)
60
>>> len(seq)
120
```

First nucleotide starts at 0

More Bioinformatics

Extracting Information from Sequences

```
>>> from string import *
>>> seq[0] + seq[1] + seq[2]
'agc'
>>> seq[0:3]
'agc'
>>> seq[3:6]
'gcc'
>>> count(seq, 'a')
35
>>> count(seq, 'c')
21
>>> count(seq, 'g')
44
>>> count(seq, 't')
20
>>> long = len(seq)
>>> nb_a = count(seq, 'a')
>>> float(nb_a) / long * 100
29.166666666666668
```

Binds additional built-in functions for strings

Find the first codon from the sequence

get 'slices' from strings:

How many of each base does this sequence contain?

Count the percentage of each base on the sequence.

More Fun with DNA Sequences

```
>>> from string import *
>>> dna =
"tgaattctatgaatggactgtccccaaagaagtaggacccactaatgcagatcctggatccctagctaagatgtattattctgctgt
gaattcgatcccactaaagat"
>>> EcoRI = "GAATTC"
>>> BamHI = 'GGATCC'
>>> EcoRI = lower(EcoRI)
>>> EcoRI
'gaattc'
>>> count(dna, EcoRI)
2
>>> find(dna, EcoRI)
1
>>> find(dna, EcoRI, 2)
88
>>> BamHI = lower(BamHI)
>>> find(dna, BamHI)
54
>>> gc=count(dna,"g")+count(dna,"c")/float(len(dna))
>>> gc
21.222222222222221
```

find and *count* are case sensitive

count returns the # of occurences of a pattern

Functions can have multiple arguments

Find(string,pattern) returns the position of the first occurrence of the pattern in the string

Find(string,pattern,n) returns the position of the nth occurrence of the pattern in the string

GC-calculation

Comment Your Code!

- How?
  - Precede comment with # sign
  - Interpreter ignores rest of the line
- Why?
  - Make code more readable by others AND yourself?
- When?
  - When code by itself is not evident

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

  - Need to say something but PL cannot express it

```
percentage = (minute * 100) / 60 # FIX: handle float division
```

Please do not over do it →

```
X = 5  #  Assign 5 to x
```