同济大学 TONGJI UNIVERSITY

生物信息学系 DEPARTMENT OF BIOINFORMATICS

Section 4

Python: Getting Started

Yong Zhang, Ph. D

School of Life Science and Technology
Tongji University

Mar 21, 2019

yzhang@tongji.edu.cn

Story: Calculating the ΔG of ATP hydrolysis

- The hydrolysis of one phosphodiester bond from ATP results in a standard Gibbs energy (ΔG^0) of –30.5 kJ/mol.
- The real ΔG value depends on the concentration of the compounds

  $\Delta G = \Delta G^0 + RT * \ln([ADP] * [P_i] / [ATP])$

TABLE 1.1 Compound Concentration in Different Tissues.

Tissue	[ATP] [mM]	[ADP] [mM]	[P_i][mM]
Liver	3.5	1.8	5.0
Muscle	8.0	0.9	8.0
Brain	2.6	0.7	2.7

Example Python Session

```
[Yong@ ~]python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> ATP = 3.5
>>> ADP = 1.8
>>> Pi = 5.0
>>> R = 0.00831
>>> T = 298
>>> deltaG0 = -30.5
>>>
>>> import math
>>> print deltaG0 + R * T * math.log(ADP * Pi / ATP)
-28.1611541611
>>>
```

The Python Shell

```
[Yong@ ~]python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> ATP = 3.5
>>> ADP = 1.8
>>> Pi = 5.0
>>> R = 0.00831
>>> T = 298
>>> deltaG0 = -30.5
>>>
>>> import math
>>> print deltaG0 + R * T * math.log(ADP * Pi / ATP)
-28.1611541611
>>>
```

- The interactive mode of Python shell is ideal for learning and for testing pieces of code.

The Python Shell

```
>>> 3 * 4
12
>>> 12.5 / 0.5
25.0
>>> (12.5 / 0.5) * 100
2500.0
>>> 3 ** 4
81
>>> 3 ** (4 + 2)
729
```

Variables

```
[Yong@ ~]python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> ATP = 3.5
>>> ADP = 1.8
>>> Pi = 5.0
>>> R = 0.00831
>>> T = 298
>>> deltaG0 = -30.5
>>>
>>> import math
>>> print deltaG0 + R * T * math.log(ADP * Pi / ATP)
-28.1611541611
>>>
```

Variables

- The operator to assign an object to a variable name is =.
- Some words cannot be used for variable names because they have a meaning in Python.
  - *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.*
- The first character of a variable name cannot be a number.
- Variable names are case sensitive.
  - var and Var are different names.
- Most special characters, i.e. *$ % @ / \ . , [ ] ( ) {}* # are not allowed in variable names.

Importing Modules

```
[Yong@ ~]python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> ATP = 3.5
>>> ADP = 1.8
>>> Pi = 5.0
>>> R = 0.00831
>>> T = 298
>>> deltaG0 = -30.5
>>>
>>> import math
>>> print deltaG0 + R * T * math.log(ADP * Pi / ATP)
-28.1611541611
>>>
```

Importing Modules

- Modules are used to reuse code and to divide big programs into smaller parts.
- For a complete list of available functions in *math*, type:

```
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asin
h', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fs
um', 'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1
p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc'
]
>>>
```

- You can get a short explanation of each function by typing:

```
>>> help(math.log)
```

Importing Modules

- *math.log* means that the *log* object (a function) is an attribute of the *math* object (a module).
- Objects can also be imported selectively from modules:

```
>>> from math import log
>>>
```

Namespaces

- Each module has its own namespace (collection of object names).
  - The namespace of the *math* module contains the names *pi*, *sqrt*, *cos*, *log*, etc.
- The dot syntax makes it possible to avoid confusion between the namespaces of the two modules.

```
>>> pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
>>> pi = 100
>>> pi
100
>>> from math import *
>>> pi
3.141592653589793
>>>
```

Calculations

```
[Yong@ ~]python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> ATP = 3.5
>>> ADP = 1.8
>>> Pi = 5.0
>>> R = 0.00831
>>> T = 298
>>> deltaG0 = -30.5
>>>
>>> import math
>>> print deltaG0 + R * T * math.log(ADP * Pi / ATP)
-28.1611541611
>>>
```

Calculations

- Arithmetical Operations in Python.

Operator	Meaning
`a + b`	addition
`a – b`	subtraction
`a * b`	multiplication
`a/b`	division
`a ** b`	power (a^b)
`a % b`	modulo: the remainder of the division *a* / *b*
`a // b`	floor division, rounds down
`a * (b + c)`	parentheses, b + c will be done before the multiplication

Calculations

- Python distinguishes between integer and floating-point numbers:

```
>>> a = 3
>>> b = 3.0
>>> a / 2
1
>>> b / 2
1.5
>>> a / 2.0
1.5
>>>
```

Calculations

- Some important functions defined in the *math* module.

Function	Meaning
`log(x)`	natural logarithm of *x* (*ln x*)
`log10(x)`	decadic logarithm of *x* (*log x*)
`exp(x)`	natural exponent of *x* (e^x)
`sqrt(x)`	square root of *x*
`sin(x), cos(x)`	sine and cosine of *x* (*x* given in radians)
`asin(x), acos(x)`	arcsin and arccos of *x* (result in radians)

- When you are using functions, the parentheses are mandatory.

```
>>> math.sqrt(49)
7.0
```

Examples

- How to calculate the distance between two points?
  - The distance *d* between two points p_1 and p_2, the coordinates of which are (x_1, y_1, z_1) and (x_2, y_2, z_2), respectively, is given by the following equation:

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Examples

```
>>> from math import sqrt
>>>
>>> x1, y1, z1 = 0.1,  0.0, -0.7
>>> x2, y2, z2 = 0.5, -1.0,  2.7
>>> dx = x1 - x2
>>> dy = y1 - y2
>>> dz = z1 - z2
>>> dsquare = pow(dx, 2) + pow(dy, 2) + pow(dz, 2)
>>> distance = sqrt(dsquare)
>>> print distance
3.56651090003
```

Examples

- How to create your own modules?
  - A Python module is a text file ending with *.py*. You can place variables and Python code, functions, etc., there.
- For instance, you can outsource the ATP constant to a module in four steps:
  - 1. Create a new text file with a text editor.
  - 2. Give it a name ending with *.py* (e.g., *hydrolysis.py*).
  - 3. Add some code. For example, add the ATP constant
    ATP = -30.5
  - 4. Finally, import the module from the Python shell (you need to store the module file in the same directory where you started the Python shell, or in the Python library, or in a directory which is added to a special Python variable *sys.path*):
    >>> import hydrolysis

Story: Calculating the amino acid frequency

```
# insulin [Homo sapiens] GI:386828
# extracted 51 amino acids of A+B chain
insulin = "GIVEQCCTSICSLYQLENYCNFVNQHLCGSHLVEALYLVCGERGFFYTPKT"

for amino_acid in "ACDEFGHIKLMNPQRSTVWY":
    number = insulin.count(amino_acid)
    print amino_acid, number
```

Comments

```
# insulin [Homo sapiens] GI:386828
# extracted 51 amino acids of A+B chain
insulin = "GIVEQCCTSICSLYQLENYCNFVNQHLCGSHLVEALYLVCGERGFFYTPKT"

for amino_acid in "ACDEFGHIKLMNPQRSTVWY":
    number = insulin.count(amino_acid)
    print amino_acid, number
```

- Comments are documentation inside the program used to describe what the code does.
- The text you want the interpreter to ignore must be preceded by the # symbol.

String variables

```
# insulin [Homo sapiens] GI:386828
# extracted 51 amino acids of A+B chain
insulin = "GIVEQCCTSICSLYQLENYCNFVNQHLCGSHLVEALYLVCGERGFFYTPKT"

for amino_acid in "ACDEFGHIKLMNPQRSTVWY":
    number = insulin.count(amino_acid)
    print amino_acid, number
```

String variables

- Strings in Python need to be enclosed by single ('abc'), double ("abc"), or triple ("'abc"' or """abc""") quotes.
- Triple-quoted strings can span multiple lines:

```
>>> text = '''Insulin is a protein produced in the pancreas.
The protein is cut proteolytically.
Its deficiency causes diabetes.'''
>>> print text
Insulin is a protein produced in the pancreas.
The protein is cut proteolytically.
Its deficiency causes diabetes.
```

String variables

- By numerical indices in square brackets, you can extract characters in certain positions. The first character is treated as in position zero:

```
>>> 'Protein'[0]
'P'
>>> 'Protein'[1]
'r'
>>> 'Protein'[-1]
'n'
>>> 'Protein'[-2]
'i'
```

String variables

- By introducing a colon in the square brackets, you can address parts of the string:

```
>>> 'Protein'[0:3]
'Pro'
>>> 'Protein'[1:]
'rotein'
```

String variables

- Strings in Python can be added using the plus (+) operator, and can also be multiplied with integer numbers:

```
>>> 'Protein' + ' ' + 'degradation'
'Protein degradation'
>>> 'Protein' * 2
'ProteinProtein'
>>> '*' * 20
'********************'
```

String variables

- The len() function returns the length of a string in number of characters:

```
>>> len('Protein')
6
```

- The s.count() function counts how often a character or a short sequence occurs in a string:

```
>>> 'protein'.count('r')
1
```

Loops with *for*

```
# insulin [Homo sapiens] GI:386828
# extracted 51 amino acids of A+B chain
insulin = "GIVEQCCTSICSLYQLENYCNFVNQHLCGSHLVEALYLVCGERGFFYTPKT"

for amino_acid in "ACDEFGHIKLMNPQRSTVWY":
    number = insulin.count(amino_acid)
    print amino_acid, number
```

Loops with *for*

- The general syntax of *for* loops is:

```
for <index variable> in <sequence>:
    <command 1>
    <command 2>
    ...
    <command x>
```

- *<sequence>* may be a string ("ACDEFGHIKL") or a collection of objects such as a list ([1, 2, 3]).
- *<index variable>* is a variable name that takes the value of the elements of *<sequence>* as it runs over its contents.
- *<command 1> … <command x>* are executed during each round of the loop.

Loops with *for*

- Running a loop over a list of numbers:

```
>>> for i in [1, 2, 3, 4, 5]:
...     print i,
...
1 2 3 4 5
>>> for i in range(10):
...     print i,
...
0 1 2 3 4 5 6 7 8 9
```

Indentation

```
# insulin [Homo sapiens] GI:386828
# extracted 51 amino acids of A+B chain
insulin = "GIVEQCCTSICSLYQLENYCNFVNQHLCGSHLVEALYLVCGERGFFYTPKT"

for amino_acid in "ACDEFGHIKLMNPQRSTVWY":
    number = insulin.count(amino_acid)
    print amino_acid, number
```

Indentation

- Indentation is used in Python to mark blocks of code that are executed together.
- Blocks of code occur in loops, conditional statements, functions, and classes.
- A block of code is initiated by a colon character and followed by the indented instructions of the block.
- All instructions of a block must be indented by the same number of spaces (it is advised to use four spaces).

Printing to the screen

```
# insulin [Homo sapiens] GI:386828
# extracted 51 amino acids of A+B chain
insulin = "GIVEQCCTSICSLYQLENYCNFVNQHLCGSHLVEALYLVCGERGFFYTPKT"

for amino_acid in "ACDEFGHIKLMNPQRSTVWY":
    number = insulin.count(amino_acid)
    print amino_acid, number
```

Printing to the screen

- You can print a value:

```
>>> print 7
7
>>> print 'insulin sequence:'
insulin sequence:
>>> sequence = 'MALWMRLLPLLALLALWGPDPAAA'
>>> print sequence
MALWMRLLPLLALLALWGPDPAAA
```

- You can print multiple values in one command:

```
>>> print 7, 'insulin sequence:', sequence
7 insulin sequence: MALWMRLLPLLALLALWGPDPAAA
```

Printing to the screen

- By default, a line break (i.e., an end-of-line character) is added after each *print* statement:

```
>>> print 7, 'insulin sequence:'; print sequence
7 insulin sequence:
MALWMRLLPLLALLALWGPDPAAA
```

- Adding a comma at the end of the last item in the *print* statement suppresses the line break.

```
>>> print 7, 'insulin sequence:',; print sequence
7 insulin sequence: MALWMRLLPLLALLALWGPDPAAA
```

Printing to the screen

- Some characters need to be replaced by *escape characters*. For example, you need to write tabulators as \t, newline characters as \n, and backslashes as \\.
- You cannot use the same quotes in a string as the ones you enclose the string in.

```
>>> print 'a single-quoted string may contain "".'
a single-quoted string may contain "".
>>> print "a double-quoted string may contain ''."
a double-quoted string may contain ''.
>>> print '''a triple-quoted string may contain '' and "".'''
a triple-quoted string may contain '' and "".
```

Examples

- How to create a random DNA sequence?

```
import random

alphabet = "AGCT"
sequence = ""
for i in range(10):
    index = random.randint(0, 3)
    sequence = sequence + alphabet[index]

print sequence
```

Examples

- How to run a sliding window over a sequence?

```
seq = "PRQTEINSEQWENCE"

for i in range(len(seq)-4):
    print seq[i:i+5]
```

Summary

- Managing Your Biological Data with Python
  - Chapter 1. The Python Shell
  - Chapter 2. Your First Python Program
- Python codes in https://bitbucket.org/krother/python-for-biologists/src/