University of Zurich[UZH]

URPP Evolution in Action

URPP tutorial

Python – basics

Dr. Heidi E.L. Lischer
University of Zurich
Switzerland

16 March, 2015

Tutorial overview

- Basics
- Control Flow
- Lists
- Input and output

Basics

- Install python:
  - Ubuntu: if not already installed `sudo apt-get install python`
  - Windows: download from www.python.org/getit/windows/
  - Mac: download from www.python.org/downloads/mac-osx/
- Start python: type **`python`** in terminal
- A simple interpreted language
  → no separate compilation step (like R)

Python 2:

```
$ python
>>> print 1 + 2
3
>>> print 'charles' + 'darwin'
charlesdarwin
```

Python 3:

```
$ python
>>> print (1 + 2)
3
>>> print ('charles' + 'darwin')
charlesdarwin
```

Basics

- Put commands in a file and execute that

```
print 1 + 2
print 'charles' + 'darwin'
```

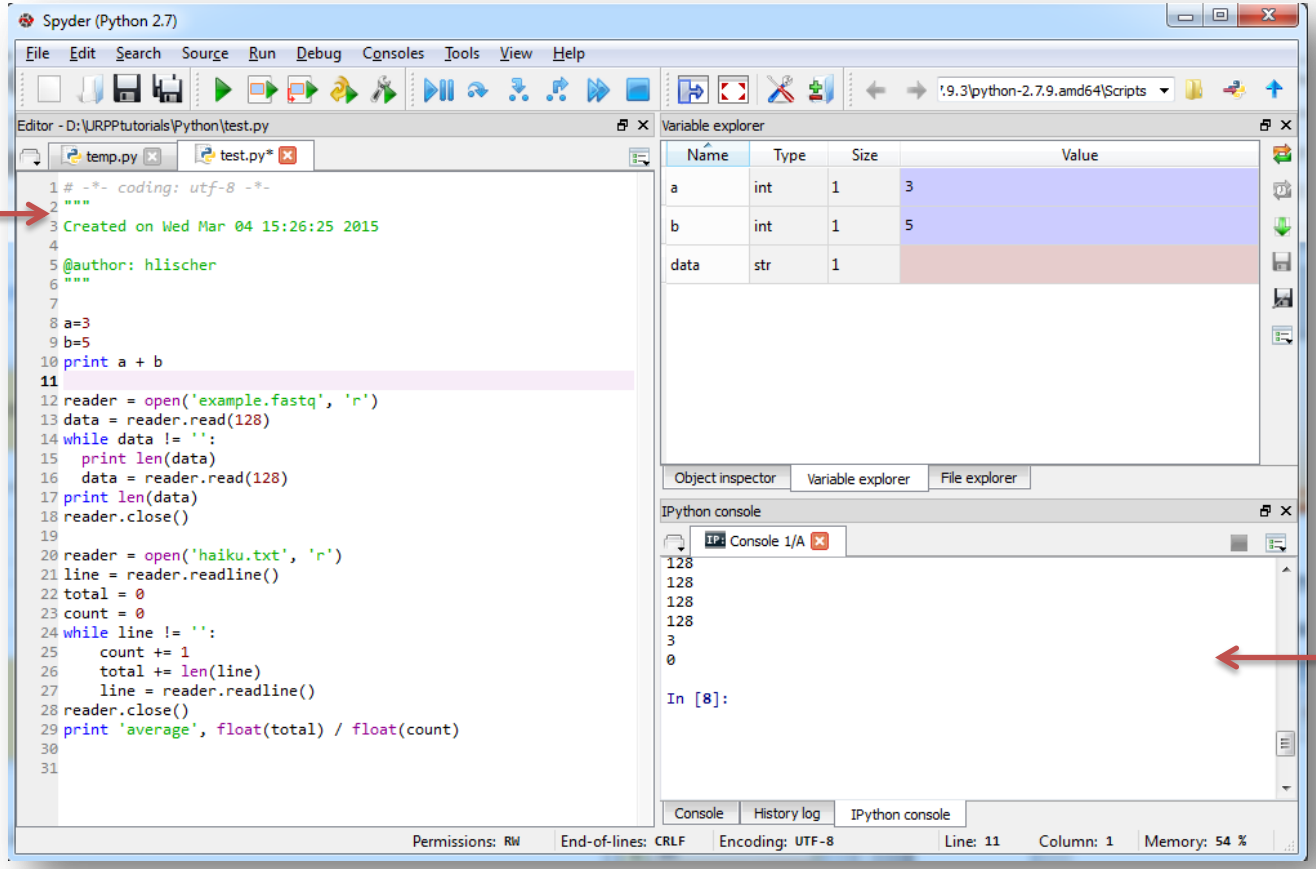
→ simple.py

```
>>> exit()
$ python simple.py
3
charlesdarwin
```

IDE

- Use an integrated development environment (IDE)
  - E.g.: Spyder (https://pythonhosted.org/spyder/index.html)

Source file

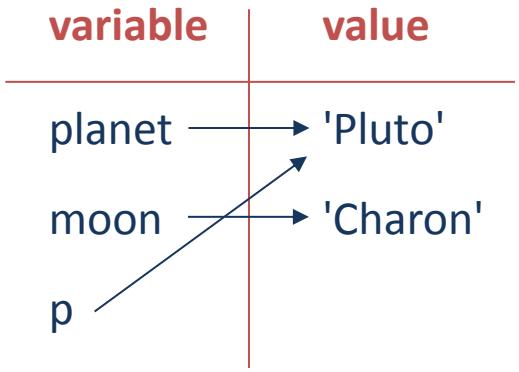


Execution shell

Variables

- **Variables:** names for values

```
>>> planet = 'Pluto'
>>> print planet
Pluto
>>> moon = 'Charon'
>>> p = planet
>>> print p
Pluto
```



- Must assign a value to a variable → else there will be an error

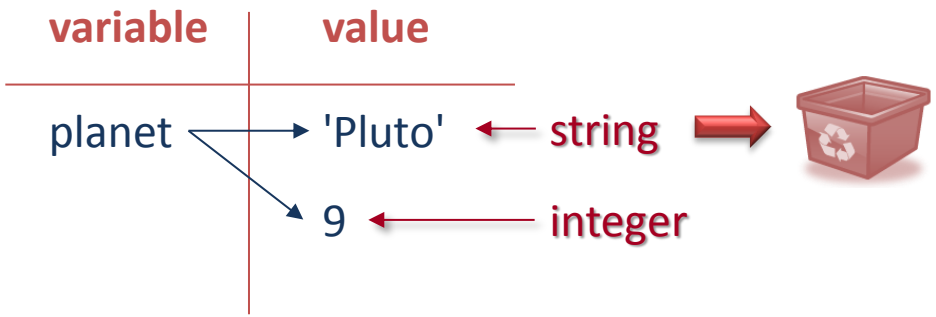
```
>>> print plant #not defined
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'plant' is not defined
```

Comment:
everything after # is ignored

Variales

- Variable is just a name
  → does not have a type

```
>>> planet = 'Pluto'
>>> planet = 9
```



→ Values are garbage collected:
If nothing refers to data any longer, the memory is recycled

Variables

- Values do have types

```
>>> string = 'two'
>>> number = 3
>>> print string * number  #repeated concatenation
twotwotwo
>>> print string + number
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

- Functions can be used to **convert between types**

```
>>> print int('2') + 3
5
>>> print 'bla' + str(3)
bla3
```

→ **`int()`**: converts string of digits to an integer

→ **`str()`**: converts a number to a string

Numbers

- There exist several types of numbers in Python:

14	32-bit integer
14.0	64-bit float

→ Convert to float: `float()`

```
>>> num = 3
>>> print float(num)
3.0
```

- Usual arithmetic operations:

Addition	+	35 + 22 'Py' + 'thon'	57 'Python'
Subtraction	-	35 - 22	13
Multiplication	*	3 * 2 'Py' * 2	6 'PyPy'
Division	/	3.0 / 2 3 / 2	1.5 1
Exponentiation	**	2 ** 0.5	1.41421356...
Remainder	%	13 % 5	3

Numbers

- Python allows you to use in-place forms of binary operators
  → make your programs more readable

```
>>> years = 500
>>> years += 1      #the same as years = years + 1
>>> print years
501
>>> years %= 10     #the same as years = years % 10
>>> print years
1
```

Comparisons

- Comparisons turn numbers or strings into **True** or **False**

Sign	Meaning	Example	Result
<	less than	3 < 5	True
>	bigger than	3 > 5	False
!=	not equal	3 != 5	True
==	equal	3 == 5	False
>=	bigger than or equal	3 >= 5	False
<=	less than or equal	3 <= 5	True

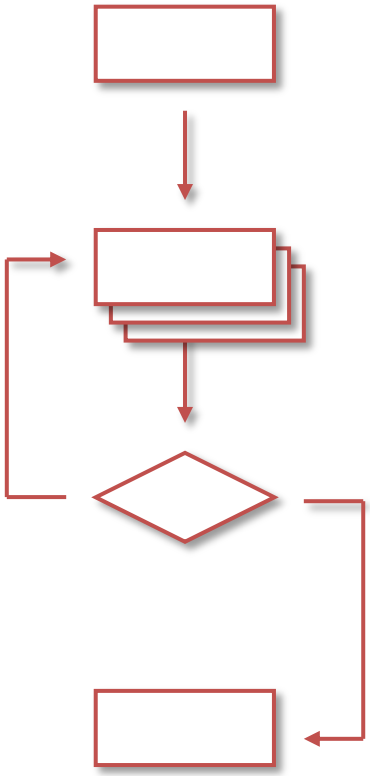
- Logical operators

Sign	Meaning	Example	Result
and	and	3 < 5 and 6 < 5	False
or	or	3 > 5 or 6 < 5	True
not	not equal	not(3 < 5)	False

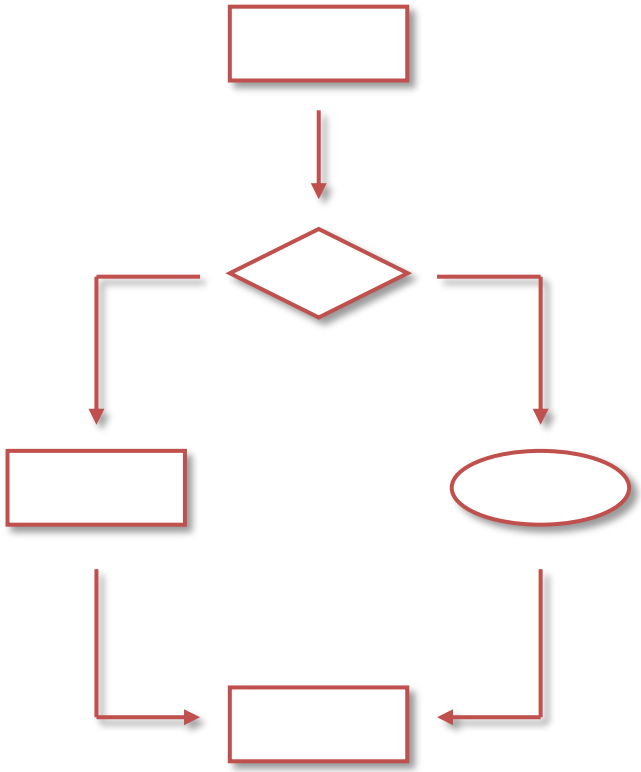
Flow control

- Real power of programs comes from:

repetition



selection



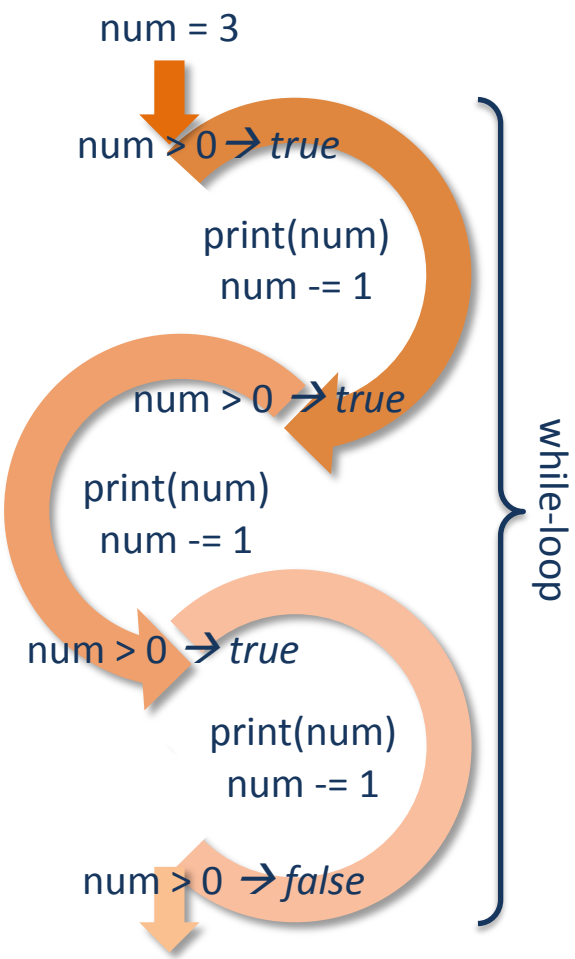
While loop

- Simplest form of repetition is while loop
  - does something as long as some condition is true

```
>>> num = 3
>>> while num > 0:
...     print num
...     num -= 1
3
2
1
```

test

do → everything which is indented



  - May also be executed zero times

```
>>> num = -3
>>> while num > 0:
...     print num
...     num -= 1
```

While loop

Why indentation?

- Studies show that's what people actually pay attention to
- Doesn't matter how much you use
  → but whole block must be consistent
- Python Style Guide (PEP 8) recommends 4 spaces
- Do not use tabs
  → different editors display tab characters with different widths

If, elif and else

- Make choices: Use if, elif, and else

```
>>> moons = 3
>>> if moons < 0:
...   print 'less'
... elif moons == 0:
...   print 'equal'
... else:
...   print 'greater'
greater
```

- Can have any number of `elif` clauses (including none)
- `else` clause is optional
- Always tested in order
  → if one test is true, its block of statements is executed and no other branch is tested

If, elif and else

- Blocks of code may contain other blocks:

```
>>> num = 0
>>> while num <= 10:
...   if (num % 2) == 1:
...     print num
...   num += 1
1
3
5
7
9
```

Print odd numbers

Count from 0 to 10

- Better way to do it:

```
>>> num = 1
>>> while num <= 10:
...   print num
...   num += 2
```

→ More efficient

Common patterns in programming

1. Writing a simple program that works
2. Tweaking it to make it more efficient

→ Write programs top-down, solving one problem at a time

Example: print primes less than 1000

```
num = 2
while num <= 1000:
    # figure out if num is prime...
    if is_prime:
        print num
    num += 1
```

Cannot be evenly divided by any other integer

```
(num % trial) == 0
```

```
is_prime = True
trial = 2
while trial < num:
    if #num divisible by trial... :
        is_prime = False
    trial += 1
```

Print primes less than 100

- Put everything together:

```
num = 2
while num <= 1000:
    is_prime = True
    trial = 2
    while trial < num:
        if (num % trial) == 0:
            is_prime = False
        trial += 1
    if is_prime:
        print num
    num += 1
```

Lists

- Collections let us store many values together
  - Most popular in Python: list

- Create a list: `[value, value, ...]`
- Get/set values: `var[index]`

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> print gases
['He', 'Ne', 'Ar', 'Kr']

>>> print gases[1]
Ne

>>> print gases[4]
IndexError: list index out of range
```

0 based indexing!

Error: try to access a list element that doesn't exist

Lists

- Get length of list: `len(list)`

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> print len(gases)
4
```

- Get values from the end of the list

```
>>> print gases[len(gases)-1]
Kr
```

  – negative indices: count from the end of the list → less error prone

```
>>> print gases[-1]
Kr
>>> print gases[-2]
Ar
```

Get last element of the list

Lists

- Get more than one element from list: `list[start:end]`

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> print gases[1:3]
['Ne', 'Ar']
```

Start is inclusive End is exclusive

Lists

- List are
  - **Mutable:** can change it after it is created

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> gases[3] = 'H'
>>> print gases
['He', 'Ne', 'Ar', 'H']

>>> gases[4] = 'Xe'
IndexError: list assignment index out of range
```

Locations must exist before assignment

  - **Heterogeneous:** can store values of different types

```
>>> helium = ['He', 2]
>>> neon = ['Ne', 8]

>>> gases = [helium, neon]
>>> print gases
[['He', 2], ['Ne', 8]]
```

Lists containing a string and an integer

Can even store references to other lists

Lists

- **Delete** entire entries: `del var[index]` → will shorten the list

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> del gases[0]
>>> print gases
['Ne', 'Ar', 'Kr']
>>> del gases[1]
>>> print gases
['Ne', 'Kr']
```

- **Add** elements: `var.append(arg)` → will extend the list

```
>>> gases.append('He')
>>> print gases
['Ne', 'Kr', 'He']
```

Append is a *method* from list

Lists

- Some useful list methods:
  - Count occurrence of an element: `var.count(arg)`

```
>>> gases = ['He', 'He', 'Ar', 'Kr']
>>> print gases.count('He')
2
```

  - Get index of first occurrence of an element: `var.index(arg)`

```
>>> print gases.index('Ar')
2
```

  - Insert an element at a given index: `var.insert(index,arg)`

```
>>> gases.insert(1, 'Ne')
>>> print gases
['He', 'Ne', 'He', 'Ar', 'Kr']
```

Lists

- Sort elements of a list: `var.sort()`

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> print gases.sort()
None
>>> print gases
['Ar', 'He', 'Kr', 'Ne']
```

Das not return the list, just sort it!

- Reverse order of a list: `var.reverse()`

```
>>> print gases.reverse()
None
>>> print gases
['Ne', 'Kr', 'He', 'Ar']
```

Das not return the list, just reverse it!

Lists

- Check if something is in a list: `arg in var`

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> print 'He' in gases
True
>>> if 'H' in gases:
...   print 'H already in list'
... else:
...   gases.append('H')
```

Lists
- range()

- Construct a sequence of integers: `range()`

```
>>> print range(5)
[0, 1, 2, 3 ,4]

>>> print range(2, 6)
[2, 3, 4, 5]

>>> print range(0, 9, 3)
[0, 3, 6]

>>> print range(10, 0)
[]

>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> print range(len(gases))
[0, 1, 2, 3]
```

Creates a list of integers from 0 to x-1

List from x to y-1

List from x to y-1 by z

Empty list

List of all indices of a list

For loop

- For loop: iterate over each value in a list

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> for gas in gases:
...     print gas
He
Ne
Ar
Kr

>>> for i in range(len(gases)):
...     print 'index of ' + gases[i] + ' is: ' + str(i)
index of He is: 0
index of Ne is: 1
index of Ar is: 2
index of Kr is: 3
```

Input and Output

- How to save data to files?
- How to read data from files?
  - Often useful to treat a file as a sequence of lines

- Simple fastq file (example.fastq):

```
@M01106:71:000000000-A8R3K:1:1101:16475:1383 1:N:0:1
ATCCTGGGTTCGTAATAATGGCGAGGCGTGAACATGAGATCTTATTTGATCTTGCGGCTTGCTGGGCCAATGCAAGCCTGGGGG
+
3A>AAFBAFBF2EGCFFGGGGGGCEG2EGAAA35BE3AAF3FGFFFD5DDDGFBFEFEEEE1@FGG1A1?333BF3?FGGEGEC
@M01106:71:000000000-A8R3K:1:1101:16606:1409 1:N:0:1
TTATTCACCTCCGGGCATAAGCGCGTGGCGCGGCCTTTCATCTGTTCGTACAGAATGCGGCTGCGTACTTTACGCAGGAACACG
+
ABAB?FFFFFFAECCGEGGGGGGEGEEEGEGGGC?AFGHGHHGHDGHGBFGFHH3FGHGCEFGGG?AEGHEHFEGEECGFHHHF
@M01106:71:000000000-A8R3K:1:1101:15510:1415 1:N:0:1
ATGATGATCAGGAGAAATTTAGCAAAAGACCGTTTCACGGTACCAAAAACACAGGCGGGAACCAATCGGGTCATTCATCTTATT
+
ABCCCFFFFFFFGGFGGGGGGGHHGHHHGHHGFGHHHHHGGHGHGHHHHGHHHHHGGGE@E1FAG33?EE1/B4F4B4B4GB4G
@M01106:71:000000000-A8R3K:1:1101:16171:1437 1:N:0:1
GTAGAGTACGCAGCTAATTGCTGCCAGCGTTAATGCATTGGCACCGGCATCCCATTGCGAAGCAATACTCCATGCGCCAATCAT
+
AAAA?FFFFABBGGGFGGGGGGHHHHHGGGGGGGHHCFHGG3GFFFGEEGEFHFGHHFFGGEGHFHHHHHHHGHED?EDFFHFH
```

Input

- How many characters are in the file?
  - Assume at the moment that each character is stored in one byte

Filename Read from file

```
reader = open('example.fastq', 'r')
data = reader.read()
reader.close()
print len(data)
899
```

Creates a file object
Read the entire content of the file
Close file object (good practice)
Get number of bytes

→ Read the entire files into memory

→ For large files it is better to read it in parts

Input

- Read in parts:

```
reader = open('example.fastq', 'r')
data = reader.read(128)
while data != '':
  print len(data)
  data = reader.read(128)
print len(data)
reader.close()
128
128
128
128
128
128
128
3
0
```

Read (max) 128 bytes
→ empty string if nothing left

Should be 0
→ because it loops until it is empty

→ More common to read a file line by line!

Input

- Read line by line: `var.readline()`

```
reader = open('example.fastq', 'r')
line = reader.readline()
while line != '':
  print line
  line = reader.readline()
reader.close()
@M01106:71:000000000-A8R3K:1:1101:16475:1383 1:N:0:1

ATCCTGGGTTCGTAATAATGGCGAGGCGTGAACATGAGATCTTATTTGATCTTGCGG...
...
```

Read a single line

Loop until no more lines in file

- Read all lines at once: `var.readlines()`

```
reader = open('example.fastq', 'r')
lines = reader.readlines()
reader.close()
print len(lines)
16
```

Stores all lines in the file as a list of strings

Input

- If memory allows:
  Read lines as list and then loop over list

```
reader = open('example.fastq', 'r')
lines = reader.readlines()
reader.close()
for line in lines:
  print len(line)
53
85
2
85
...
```

Loop over lines in the file

Prints length of the line including newline character!

Input

- Remove newline character at end of line: `line.rstrip('\r\n')`

```
reader = open('example.fastq', 'r')
lines = reader.readlines()
reader.close()
for line in lines:
  line = line.rstrip('\r\n')
  print len(line)
52
84
1
...
```

- Remove any whitespace at the start and end of line: `line.strip()`

```
example = '   hello world   '
print example.strip()
hello world
```

Output

- Write data in files:
  - `write()`
  - `writelines()`

```
gases = ['He', 'Ne', 'Ar', 'Kr']
writer = open('tmp.txt', 'w')
writer.write('Gas list: ')
writer.writelines(gases)
writer.close()
```

Creates a file object (same function)
→ `'w'`: write to file
→ Overwrites file if it already exist

Use `write` to write a string to the file

Use `writelines` to write each string in a list

  - tmp.txt:

```
Gas list: HeNeArKr
```

All on same line:
Didn't write any end-of-line characters ('\n')
→ Python only writes what you tell

Output

- Often simpler to use `print >>`
  - Automatically adds a newline

```
gases = ['He', 'Ne', 'Ar', 'Kr']
writer = open('tmp.txt', 'w')
print >> writer, 'Gas list:'
for gas in gases:
  print >> writer, gas
writer.close()
```

Specify file object after `print >>`, followed by the thing you like to print

  - tmp.txt:

```
Gas list:
He
Ne
Ar
Kr
```

Acknowledgment

- Sources:
  - **http://software-carpentry.org/v4/python/index.html**
  - http://pythonforbiologists.com/

- Further reading:
  - http://www.pasteur.fr/formation/infobio/python/
  - http://www.programmingforbiologists.org/