Programming in Python

MBV-INFx410

Fall 2016

Who am I?

- Karin Lagesen, bioinformatician at the Veterinary Institute, Oslo
- National biomedical research Institute, centering on animal and fish health, and food safety
- karin.lagesen@vetinst.no

Programs

- Everything that happens on a computer is done by a program
- A program is a set of instructions that tells the computer exactly what to do
- Computers very literal – only do what the program tells them to do

Why program?

- Increases flexibility: can analyze what you want, not just what somebody else thought was smart
- Solve problems such as
  - Files being in the wrong format
  - Extract subsets of data
  - Modify data according to criteria
  - Chain other programs together in a pipeline

How to program

- Program: ordered set of instructions
- Programming can be compared to a:
  - Cooking recipe
  - Ikea furniture instructions
  - Lab protocol
- Programming language: instruction set

How to make a program

- Need a programming language
- Programming language dictates the set of available instructions
- Several types of languages – several types of instruction sets
- Some languages require the program to be compiled before running, others interpreted on the fly

Interpreted languages

- Program interpreted "on-the-fly"
- Programs often called scripts
- Example of interpreted languages:
  - General purpose: perl, python
  - Special purpose: R
- Possible disadvantage: can be slower than compiled programs.

About python

- General purpose programming language, created in 1991
- Goal: be very concise and enable clear programming
- Designed to be very readable because
  - Easier to find bugs
  - Easier to understand later
  - Easier to maintain
  - Easier to learn

Interactive vs. batch mode

- Python can be used interactively
- Useful for testing etc
- Most common: save code in text file, run with python
- Called batch mode

Interactive shell example

```
[karinlag@freebee]~% module load python2

[karinlag@freebee]~% python
Python 2.7.10 (default, Jul  1 2015, 11:02:23)
[GCC Intel(R) C++ gcc 4.4 mode] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print 2+2
 4
 >>> print 2*6
 12
 >>> print "Hello World"
 Hello World
 >>> print len("Hello World")
 11
 >>>
```

Task: log in to freebee, run this example.

Creating script

- A script is code in a file which is run
- Create script file (use nano etc)
  - Insert different calculations
  - Save the file as first.py
- Run script
- Note: in script have to use print to get results
- Print more things: use comma between elements

first.py

- Open nano
- Input text below into file, save file as first.py
- Run file like this: python first.py

```
print 2+2
print 4-2
print 3*6
print 2**8
```

Text found in script file

```
4
2
18
256
```

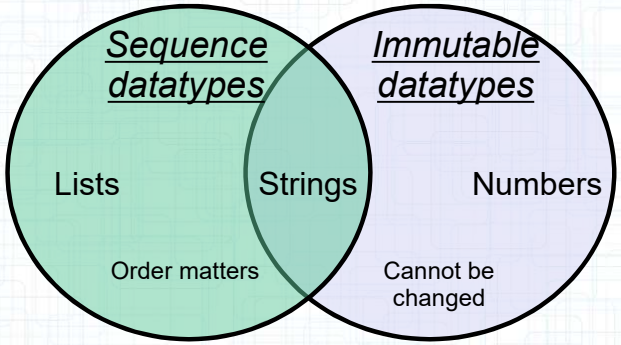
Result when running it

Python data types

- Data type: the different kinds of data that python can deal with
- **Numbers**: integers and floats (decimal numbers)
- **Strings**: text
- **Lists**: ordered collection of elements
- **Dictionaries**: mapping elements

- Python has additional types not discussed in this course

Two different type features



Python operators

TABLE 2.1:
Arithmetic-Style Operators

Symbol	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
%	Modulus (remainder)

Dealing with numbers

```
>>> print 2+2
4
>>> print 4-2
2
>>> print 5*23
115
>>> print 12/6
2
>>> print 11/6
1
>>> print 11.0/6
1.8333333333333333
>>>
```

Integer division rounds downwards, need to make sure one is a decimal number to get a decimal answer.

Strings

- Used to keep text
- Use ', " or ''' to delineate
- Remember: same type on each end
- Newline: \n
- Tab: \t

String operations

- "string".replace("X", "Y")
  - Replaces all Xes with Ys in string
- "string".count("X")
  - Counts all occurrences of X in string
- "string".find("X")
  - Reports where the first occurrence of X is in string – zero based
- "string".split("character")
  - Splits string on all occurrences of X in string, default is whitespace (tab, space)
- Concatenate: word1 + word2

String examples

```
>>> print "Abracadabra".replace("a","X")
'AbrXcXdXbrX'
>>> print "Abracadabra".count("a")
4
>>> print "Abracadabra".find("a")
3
>>> print "Abracadabra".split("a")
['Abr', 'c', 'd', 'br', '']
>>> print “Abracadabra” + “AliBaba”
 'AbracadabraAlibaba'
```

String questions

- Have string "ATG,GTC,GGC"
- How do you do the following:
  - Count how many Gs the string contains?
  - Replace all Ts with Us
  - Split the string on commas

```
>>> print "ATG,GTC,GGC".count("G")
4
>>> print "ATG,GTC,GGC".replace("T", "U")
'AUG,GUC,GGC'
>>> print "ATG,GTC,GGC".split(",")
['ATG', 'GTC', 'GGC']
>>>
```

Variables

- A variable is used to carry data in a program
- Naming variables:
  - Letters, numbers and _
  - CasE sEnsitive
  - Numbers may not be first
  - Some words are reserved
  - Convention: small letters, underscore to separate words

Reserved words

```
and       del       from      not       try
as        elif      global    None      while
assert    else      if        or        with
break     except    import    pass      yield
class     exec      in        print
continue  finally   is        raise
def       for       lambda    return
```

Using variables

Variable assignment: giving a variable a value, i.e. specifying the content of the variable

```
>>> a = 2
>>> b = 3
>>> print a*b
6
>>> print a = "Hello"
>>> print b = "World"
>>> print a, b
Hello World
>>>
```

- We are using the variable instead of the string or number itself
- Can do the same thing to another string or number

Dynamic, strong typing

```
>>> a = 2
>>> print a
2
>>> print a/2
1
>>> a = "Hello"
>>> print a
Hello
>>> print a/2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>>
```

- Dynamic: no need to specify type
- Strong: python objects if we do something a type does not support

Lists

- Ordered collection of elements

```
>>> list1 = [1, "2", "hello", 4]
```

- Can hold elements of any type, including another list

```
>>> list1 = ["a", "c", "b"]
>>> list2 = ["X", "Y", list1]
>>> print list2
['X', 'Y', ['a', 'c', 'b']]
```

Lists: created by having comma between elements, and having [] around them

- Can be sorted (in place) using .sort()

```
>>> list1.sort()
>>> print list1
['a', 'b', 'c']
```

Adding to list

- Create empty list:
  - list1 = []
- Create list by splitting string on char:
  - list1 = string.split("character")
- Add element to end of list:
  - list1.append(elem)
- Insert in list at specific position:
  - list1.insert(pos,elem)

List adding example

```
>>> list1 = "A,B,C,D,E".split(",")
>>> print list1
['A', 'B', 'C', 'D', 'E']
>>> list1.append('F')
>>> print list1
['A', 'B', 'C', 'D', 'E', 'F']
>>> list1.insert(3,'G')
>>> print list1
['A', 'B', 'C', 'G', 'D', 'E', 'F']
```

Take text, split, thereby creating list

Adding to list

Insert new element in list

List removal

- Remove specified element
  - list1.remove(elem)
- return elem in index, default is last
  - list1.pop(index)

Input from command line

- Input is stored in list called sys.argv
- Everything after script name on command line is in list called sys.argv

```
[karinlag@freebee]~/teaching% cat sys_argv.py
import sys
print sys.argv
[karinlag@freebee]~/teaching% python sys_argv.py a b c
['sys_argv.py', 'a', 'b', 'c']
[karinlag@freebee]~/teaching% python sys_argv.py 1 2
['sys_argv.py', '1', '2']
[karinlag@freebee]~/teaching%
```

- To use: import sys on top

A command line calcuator

- Task: create program that takes two words in on command line, glues them together, outputs the results.
- Do: open nano, and write in the code in the box below, save as add.py
- Run: python add.py word1 word2

```
import sys

first = sys.argv[1]
second = sys.argv[2]

sum = first + second
print sum
```

Type conversions

- Everything that comes in from the command line is a string
- How to convert:
  - int(X)
    - string cannot have decimals
    - floats will be floored
  - float(X)
  - str(X)

Command line calculator cont.

- Have to convert the input to add them together
- How: newVariable = int(inputedValue)
- newVariable now contains value as an int

```
import sys

first = int(sys.argv[1])
second = int(sys.argv[2])

sum = first + second
print sum
```

Sequence methods

- Works on strings and lists
- Indexing
  - Index starts at zero
  - Negative indices go from right edge
- Slicing
  - Can access portions of sequence using indices
- In operator – test for membership
- Concatenation – add two together with +
- Len, min, max

Indices

```
>>> text = "ABCDEFG"
>>> print text[2]
'C'
>>> print text[-2]
'F'
>>> print text[2:4]
'CD'
>>> print text[2:-2]
'CDE'
>>> print text[:4]
'ABCD'
>>> print text[4:]
'EFG'
>>>
```

0	*1*	*2*	*3*	*4*	*5*	*6*
A	B	C	D	E	F	G
-7	*-6*	*-5*	*-4*	*-3*	*-2*	*-1*

REMEMBER: Python is zero-based!

: used for slicing

If nothing before/after :, shorthand for begin/end

- Note: for slicing, it is [from and including : to but excluding]

in operator

- Test if element is in sequence
- Works with lists and strings

```
>>>> X = [1,4,8,2,9]
>>> print X
[1, 4, 8, 2, 9]
>>> print 5 in X
False
>>> print 8 in X
True
>>>
```

```
>>> X = "ABCDEF"
>>> print X
'ABCDEF'
>>> print "Y" in X
False
>>> print "BC" in X
True
>>>
```

Concatenation

- Concatenation: + sign
- Can only concatenate same types

```
>>> a = [1,2]
>>> b = [3,4]
>>> print a + b
[1, 2, 3, 4]
>>> c = 56
>>> print a + c
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "int") to list
>>>
```

Len, min, max

- Len: length of sequence object
- Min: minimum
- Max: maximum

```
>>> txt1 = "ABCDEF"
>>> print len(txt1)
6
>>> print max(txt1)
'F'
>>> print min(txt1)
'A'
>>>
```

Sequence manipulation questions

- Have the list [1,4,8,2,10]
  - Find the maximum number
  - Find out if the number 9 is in the list
  - Add the number 9 to the list
  - Is number 9 in the list now?
  - Use sort to find the two lowest numbers

Sequence manipulation

```
>>> list1 = [1,4,8,2,10]
>>> print list1
[1, 4, 8, 2, 10]
>>> print max(list1)
10
>>> print 9 in list1
False
>>> list1.append(9)
>>> print list1
[1, 4, 8, 2, 10, 9]
>>> print 9 in list1
True
>>> list1.sort()
>>> print list1
[1, 2, 4, 8, 9, 10]
>>> print list1[:2]
[1, 2]
>>>
```

Define list

Max in list

Is 9 in list?

Add 9 to list

Retest for 9

Sort list

Get two lowest numbers

Dictionaries

- Stores unordered, arbitrarily indexed data
- Consists of key-value pairs
  - dict = {key:value, key:value, key:value...}
- Note: keys must be immutable!
  - ergo: numbers or strings
- Values may be anything, incl. another dictionary
- Mainly used for storing associations or mappings

Create, add, lookup, remove

- Creation:
  - mydict = {} (empty), or
  - mydict = { mykey:myval, mykey2:myval2 }
- Adding:
  - mydict[key] = value
- Lookup:
  - mydict[key]
- Remove:
  - del mydict[key]

Dictionary: created by : between key:value, comma between pairs, and {} around everything.

Dictionary methods

- All keys:
  - mylist.keys() - returns list of keys
- All values:
  - mydict.values() - returns list of values
- All key-value pairs as list of tuples:
  - mydict.items()
- Get one specific value:
  - mydict[key]
- Test for presence of key:
  - key in mydict – returns True or False

Dictionary example

```
>>> dict1 = {}
>>> dict1["georg"] = "kate"
>>> print dict1
{'georg': 'kate'}
>>> dict1["blue"] = 4
>>> print dict1
{'blue': 4, 'georg': 'kate'}
>>> dict1[5] = "red"
>>> print dict1
{'blue': 4, 5: 'red', 'georg': 'kate'}
>>> print dict1["blue"]
4
>>> print dict1.keys()
['blue', 5, 'georg']
>>> print dict1.values()
[4, 'red', 'kate']
>>> print "georg" in dict1
True
>>> print "kate" in dict1
False
```

Define dictionary

Add to dict

Lookup key 5

Show all keys

Show all values

Test if element is in dict, NOTE: only looks in keys, not values.

Dictionary questions

- Have this dictionary:
  dict1 = {"A": 1, 1:"A", "B":[1,2,3]}
- Find out the following:
  - how many key – value pairs are there?
  - add the key value pair "str": 2 to the dictionary
  - print the value that is stored with key "str"
  - Show all values in dictionary

Dictionary manipulation

```
>>> dict1 = {"A": 1, 1:"A", "B":[1,2,3]}
>>> print len(dict1)
3
>>> dict1["str"] = 2
>>> print dict1["str"]
2
>>> print dict1.values()
[1, 'A', [1, 2, 3], 2]
>>>
```

Define dictionary

Length works as before

Adding "str":2

Print the value of "str"

Print all values

What does this script do?

```
text = "ATCCGGAGGAGGA"
As = text.count("A")
Ts = text.count("T")
print "Length of", text, "is", len(text)
print "Nos of As is", As
print "Nos of Ts is", Ts
print "AT content is", (As + Ts)/float(len(text))
```

```
Length of ATCCGGAGGAGGA is 13
Nos of As is 4
Nos of Ts is 1
AT content is 0.384615384615
```

Modify: how do you calculate percentages instead?

Calculate GC content

- GCcontent.py
  - Get DNA string from command line: AGCAGATCAGCGA
  - Calculate the frequency of Gs and Cs separately in the string
  - Calculate the frequency of the dinucleotide "GC" in the input string
  - Print the results to screen

GCcontent.py

```
import sys
DNA = sys.argv[_]

# count G's, C's and dinculotide 'GC'
g = ___.count(___)
gfreq = g/float(len(___))
c = DNA.___(___)
cfreq = _/_____(len(DNA))
gc = DNA.count(____)/_____(len(___))

print "Frequency of Gs: ", ____
print "Frequency of Cs: ", _____
print "Frequency of GCs:", _______
```

Fill in the blanks!

GCcontent.py

```
import sys
DNA = sys.argv[1]

# count G's, C's and dinculotide 'GC'
g = DNA.count("C")
gfreq = g/float(len(DNA))
c = DNA.count("G")
cfreq = c/float(len(DNA))
gcfreq = DNA.count("GC")/float(len(DNA))

print "Frequency of Gs:", gfreq
print "Frequency of Cs:", cfreq
print "Frequency of GCs:", gcfreq
```

```
[karinlag@freebee]% python GCcontent.py AGCAGATCAGCGA
Frequency of Gs: 0.230769230769
Frequency of Cs: 0.307692307692
Frequency of GCs: 0.153846153846
[karinlag@freebee]~/teaching/mbvinf_2013%
```

Print open reading frame

- ORF.py
  - Get string from command line: `ATCAATGAGATTACAGAGCTAAGAC`
  - Replace all Ts with Us
  - Find position of start codon AUG
  - Find position of stop codon UAA
  - Print sequence from (including) start codon to stop codon (excluding)

ORF.py

```
import sys
seq = sys.____[1]
print "Sequence is '" + ___ + "'"

# replace all Ts with Us
seq = seq._______("T","U")

# find position of start codon AUG
start = seq.____("AUG")

# find position of stop codon UAA
stop = ___.find(_____)

# print sequence from (including) start codon
# to stop codon (excluding)
print "Sequence between start and stop is ", seq[_____:stop]
```

ORF.py

```
import sys
seq = sys.argv[1]
print "Sequence is '" + seq + "'"

# replace all Ts with Us
seq = seq.replace("T","U")

# find position of start codon AUG
start = seq.find("AUG")

# find position of stop codon UAA
stop = seq.find("UAA")

# print sequence from (including) start codon
# to stop codon (excluding)
print "Sequence between start and stop is ", seq[start:stop]
```

```
[karinlag@freebee]$ python ORF.py
Sequence is 'ATCAATGAGATTACAGAGCTAAGAC'
Sequence between start and stop is AUGAGAUUACAGAGC
[karinlag@titan mbvinf_2013]$
```