Essential Computing for Bioinformatics

Lecture 5

High-level Programming with Python

Container Objects

MARC: Developing Bioinformatics Programs
July 2009

Alex Ropelewski
PSC-NRBSC
Bienvenido Vélez
UPR Mayaguez

Essential Computing for Bioinformatics

- The following material is the result of a curriculum development effort to provide a set of courses to support bioinformatics efforts involving students from the biological sciences, computer science, and mathematics departments. They have been developed as a part of the NIH funded project "Assisting Bioinformatics Efforts at Minority Schools" (2T36 GM008789). The people involved with the curriculum development effort include:

- Dr. Hugh B. Nicholas, Dr. Troy Wymore, Mr. Alexander Ropelewski and Dr. David Deerfield II, National Resource for Biomedical Supercomputing, Pittsburgh Supercomputing Center, Carnegie Mellon University.
- Dr. Ricardo González Méndez, University of Puerto Rico Medical Sciences Campus.
- Dr. Alade Tokuta, North Carolina Central University.
- Dr. Jaime Seguel and Dr. Bienvenido Vélez, University of Puerto Rico at Mayagüez.
- Dr. Satish Bhalla, Johnson C. Smith University.

- Most recent versions of these presentations can be found at http://marc.psc.edu/

Outline

- Lecture 2 Homework
- Lists and Other Sequences
- Dictionaries and Sequence Translation
- Finding ORF's in sequences

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Lecture 2 Homework: Finding Patterns Within Sequences

```
from string import *
def searchPattern(dna, pattern):
    'print all start positions of a pattern string inside a target string'
    site = findDNAPattern (dna, pattern)
    while site != -1:
        print 'pattern %s found at position %d' % (pattern, site)
        site = findDNApattern (dna, pattern, site + 1)
```

```
>>> searchPattern("acgctaggct","gc")
pattern gc at position 2
pattern gc at position 7
>>>
```

What if DNA may contain unknown nucleotides ‘X’?

Example from Pasteur Institute Bioinformatics Using Python

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Lecture 2 Homework: One Approach

Write your own find function:

```
def findDNAPattern(dna, pattern,startPosition, endPosition):
    'Finds the index of the first ocurrence of DNA pattern within DNA sequence'
    dna = dna.lower() # Force sequence and pattern to lower case
    pattern = pattern.lower()
    for i in xrange(startPosition, endPosition):
        # Attempt to match pattern starting at position i
        if (matchDNAPattern(dna[i:],pattern)):
            return i
    return -1
```

Top-Down Design: From BIG functions to small helper functions

Lecture 2 Homework: One Approach

Write your own find function:

```
def matchDNAPattern(sequence, pattern):
    'Determines if DNA pattern is a prefix of DNA sequence'
    i = 0
    while ((i < len(pattern)) and (i < len(sequence))):
        if (not matchDNANucleotides(sequence[i], pattern[i])):
            return False
        i = i + 1
    return (i == len(pattern))
```

Top-Down Design: From BIG functions to small helper functions

Lecture 2 Homework: One Approach

Write your own find function:

```
def matchDNANucleotides(base1, base2):
    'Returns True is nucleotide bases are equal or one of them is unknown'
    return (base1 == 'x' or
            base2 == 'x' or
           (isDNANucleotide(base1) and (base1 == base2)))
```

Top-Down Design: From BIG functions to small helper functions

Lecture 2 Homework: One Approach

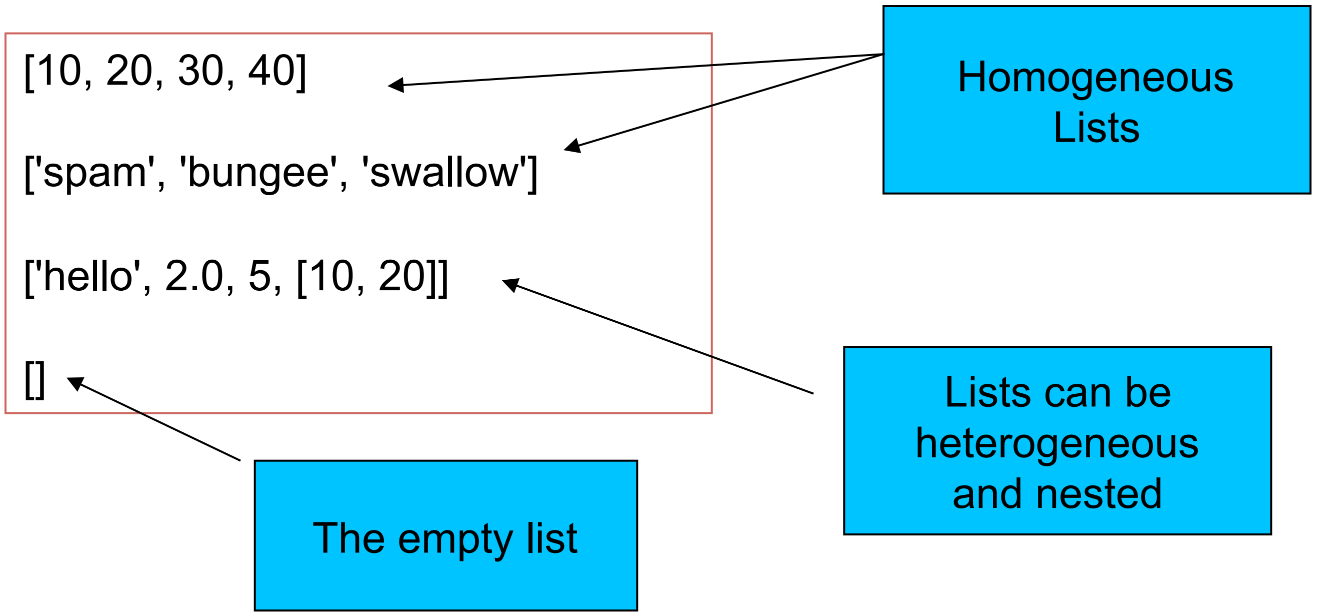
Using default parameters:

```
def findDNAPattern(dna, pattern,startPosition=0, endPosition=None):
    'Finds the index of the first ocurrence of DNA pattern within DNA sequence'
    if (endPosition == None):
        endPosition = len(dna)
    dna = dna.lower() # Force sequence and pattern to lower case
    pattern = pattern.lower()
    for i in xrange(startPosition, endPosition):
        # Attempt to match patter starting at position i
        if (matchDNAPattern(dna[i:],pattern)):
            return i
    return -1
```

Top Down Design: A Recursive Process

- Start with a high level problem
- Design a high-level function assuming existence of ideal lower level functions that it needs
- Recursively design each lower level function top-down

List Values



```
[10, 20, 30, 40]

['spam', 'bungee', 'swallow']

['hello', 2.0, 5, [10, 20]]

[]
```

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Generating Integer Lists

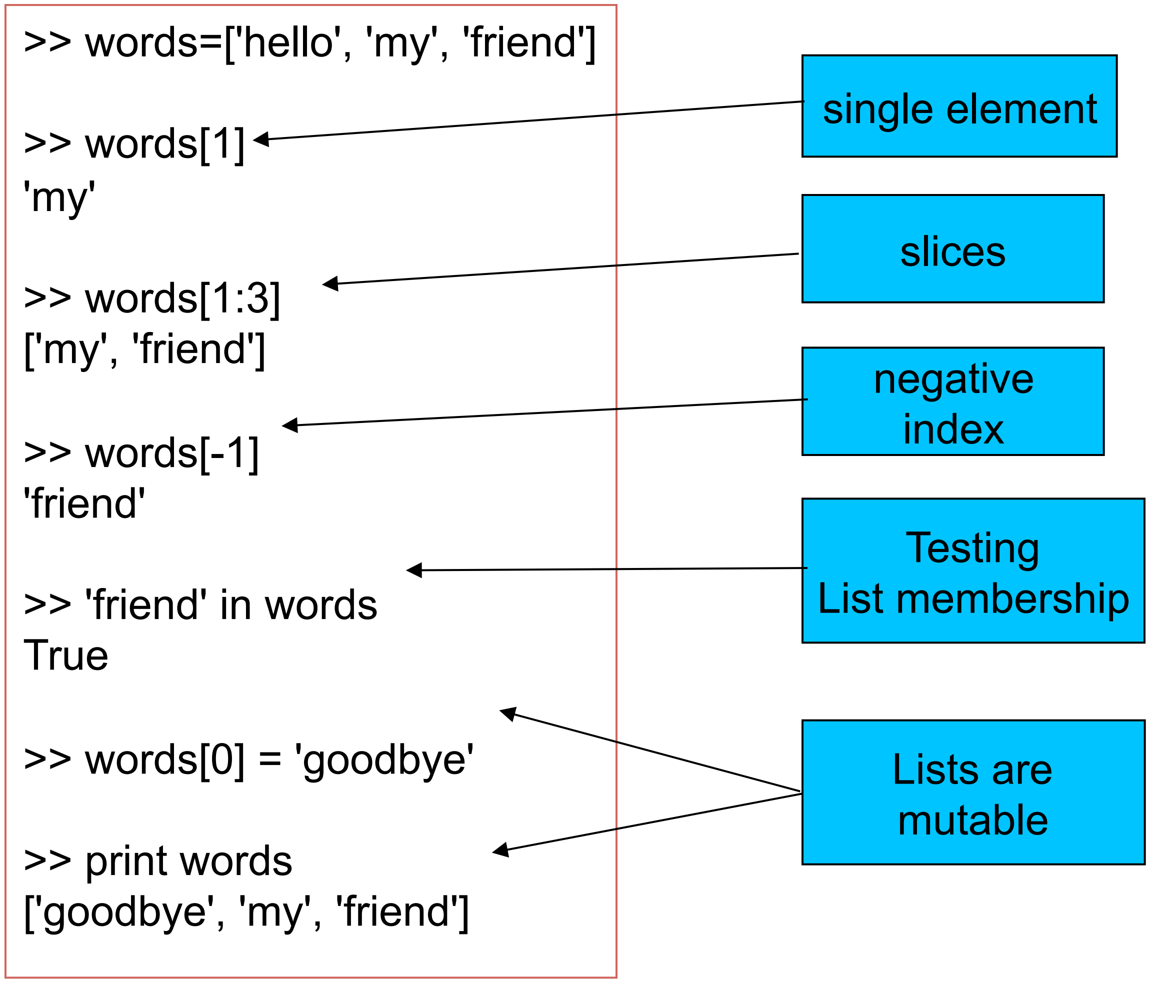
```
>>> range(1,5)

[1, 2, 3, 4]

>>> range(10)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(1, 10, 2)

[1, 3, 5, 7, 9]
```

In General

range(first,last+1,step)

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Accessing List Elements



```
>> words=['hello', 'my', 'friend']

>> words[1]
'my'

>> words[1:3]
['my', 'friend']

>> words[-1]
'friend'

>> 'friend' in words
True

>> words[0] = 'goodbye'

>> print words
['goodbye', 'my', 'friend']
```

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

More List Slices

```
>> numbers = range(1,5)

>> numbers[1:]
[1, 2, 3, 4]

>> numbers[:3]
[1, 2]

>> numbers[:]
[1, 2, 3, 4]
```

Slicing operator always returns a NEW list

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Modifying Slices of Lists

```
>>> list = ['a', 'b', 'c', 'd', 'e', 'f']

>>> list[1:3] = ['x', 'y']

>>> print list
['a', 'x', 'y', 'd', 'e', 'f']
```

Replacing slices

```
>>> list[1:3] = []

>>> print list
['a', 'd', 'e', 'f']

>>> list = ['a', 'd', 'f']
```

Deleting slices

```
>>> list[1:1] = ['b', 'c']

>>> print list
['a', 'b', 'c', 'd', 'f']

>>> list[4:4] = ['e']

>>> print list
['a', 'b', 'c', 'd', 'e', 'f']
```

Inserting slices

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Traversing Lists (2 WAYS)

```
codons = ['cac', 'caa', 'ggg']
```

```
for codon in codons:
    print codon
```

```
i = 0
while (i < len(codons)):
    codon = codons[i]
    print codon
    i = i + 1
```

Which one do you prefer? Why?

Why does Python provide both `for` and `while`?

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

String ↔ List Conversion

```
def stringToList(theString):
    'Returns the input string as a list of characters’
    result = []
    for element in theString:
        result = result + [element]
    return result

def listToString(theList):
    'Returns the input list of characters as a string’
    result = ””
    for element in theList:
        result = result + element
    return result
```

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Complementing Sequences: Utilities

```
DNANucleotides='acgt'
DNAComplements='tgca’

def isDNANucleotide(nucleotide):
    'Returns True when n is a DNA nucleotide’
    return (type(nucleotide) == type("") and
            len(nucleotide)==1 and
            nucleotide.lower() in DNANucleotides)

def isDNASequence(sequence):
    'Returns True when sequence is a DNA sequence’
    if type(sequence) != type(""):
        return False;
    for base in sequence:
        if (not isDNANucleotide(base.lower())):
            return False
    return True
```

Complementing Sequences

```
def getComplementDNANucleotide(n):
    'Returns the DNA Nucleotide complement of n’
    if (isDNANucleotide(n)):
        return(DNAComplements[find(DNANucleotides,n.lower())])
    else:
        raise Exception ("getComplementDNANucleotide: Invalid DNA
                                  sequence: " + n)

def getComplementDNASequence(sequence):
    'Returns the complementary DNA sequence’
    if (not isDNASequence(sequence)):
        raise Exception("getComplementRNASequence: Invalid DNA
                                  sequence: " + sequence)
    result = ””
    for base in sequence:
        result = result + getComplementDNANucleotide(base)
    return result
```

Complementing a List of Sequences

```
def getComplementDNASequences(sequences):
    'Returns a list of the complements of list of DNA sequences’
    result = []
    for sequence in sequences:
        result = result + [getComplementDNASequence(sequence)]
    return result
```

```
>>> getComplementDNASequences(['acg', 'ggg'])

['tgc', 'ccc']

>>>
```

Python Sequence Types

Type	Description	Elements	Mutable
StringType	Character string	Characters only	no
UnicodeType	Unicode character string	Unicode characters only	no
ListType	List	Arbitrary objects	yes
TupleType	Immutable List	Arbitrary objects	no
XRangeType	return by xrange()	Integers	no
BufferType Buffer	return by buffer()	arbitrary objects of one type	yes/no

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Operations on Sequences

Operator/Function	Action	Action on Numbers
[ ], ( ), ' '	creation	
s + t	concatenation	addition
s * n	repetition n times	multiplication
s[i]	indexation	
s[i:k]	slice	
x in s	membership	
x not in s	absence	
for a in s	traversal	
len(s)	length	
min(s)	return smallest element	
max(s)	return greatest element	

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Exercises

Design and implement Python functions to satisfy the following contracts:

- Return the list of codons in a DNA sequence for a given reading frame
- Return the lists of restriction sites for an enzyme in a DNA sequence
- Return the list of restriction sites for a list of enzymes in a DNA sequence
- Find all the ORF’s of length >= n in a sequence

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Dictionaries

Dictionaries are *mutable unordered collections* which may contain objects of different sorts. The objects can be accessed using a *key*.

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Genetic Code As Python Dictionary

```
GeneticCode =
      { 'ttt': 'F', 'tct': 'S', 'tat': 'Y', 'tgt': 'C',
        'ttc': 'F', 'tcc': 'S', 'tac': 'Y', 'tgc': 'C',
        'tta': 'L', 'tca': 'S', 'taa': '*', 'tga': '*',
        'ttg': 'L', 'tcg': 'S', 'tag': '*', 'tgg': 'W',
        'ctt': 'L', 'cct': 'P', 'cat': 'H', 'cgt': 'R',
        'ctc': 'L', 'ccc': 'P', 'cac': 'H', 'cgc': 'R',
        'cta': 'L', 'cca': 'P', 'caa': 'Q', 'cga': 'R',
        'ctg': 'L', 'ccg': 'P', 'cag': 'Q', 'cgg': 'R',
        'att': 'I', 'act': 'T', 'aat': 'N', 'agt': 'S',
        'atc': 'I', 'acc': 'T', 'aac': 'N', 'agc': 'S',
        'ata': 'I', 'aca': 'T', 'aaa': 'K', 'aga': 'R',
        'atg': 'M', 'acg': 'T', 'aag': 'K', 'agg': 'R',
        'gtt': 'V', 'gct': 'A', 'gat': 'D', 'ggt': 'G',
        'gtc': 'V', 'gcc': 'A', 'gac': 'D', 'ggc': 'G',
        'gta': 'V', 'gca': 'A', 'gaa': 'E', 'gga': 'G',
        'gtg': 'V', 'gcg': 'A', 'gag': 'E', 'ggg': 'G'
      }
```

A Test DNA Sequence

```
cds ='''atgagtgaacgtctgagcattaccccgctggggccgtatatcggcgcacaaa
tttcgggtgccgacctgacgcgcccgttaagcgataatcagtttgaacagctttaccatgcggtg
ctgcgccatcaggtggtgtttctacgcgatcaagctattacgccgcagcagcaacgcgcgctggc
ccagcgttttggcgaattgcatattcaccctgtttacccgcatgccgaaggggttgacgagatca
tcgtgctggatacccataacgataatccgccagataacgacaactggcataccgatgtgacattt
attgaaacgccacccgcaggggcgattctggcagctaaagagttaccttcgaccggcggtgatac
gctctggaccagcggtattgcggcctatgaggcgctctctgttcccttccgccagctgctgagtg
ggctgcgtgcggagcatgatttccgtaaatcgttcccggaatacaaataccgcaaaaccgaggag
gaacatcaacgctggcgcgaggcggtcgcgaaaaacccgccgttgctacatccggtggtgcgaac
gcatccggtgagcggtaaacaggcgctgtttgtgaatgaaggctttactacgcgaattgttgatg
tgagcgagaaagagagcgaagccttgttaagtttttgtttgcccatatcaccaaaccggagttt
caggtgcgctggcgctggcaaccaaatgatattgcgatttgggataaccgcgtgacccagcacta
tgccaatgccgattacctgccacagcgacggataatgcatcgggcgacgatccttggggataaac
cgttttatcgggcggggtaa'''.replace('\n','').lower()
```

CDS Sequence -> Protein Sequence

```
def translateDNASequence(dna):
    if (not isDNASequence(dna)):
        raise Exception('translateDNASequence: Invalid DNA sequence’)
    prot = ””
    for i in range(0,len(dna),3):
        codon = dna[i:i+3]
        prot = prot + GeneticCode[codon]
    return prot
```

```
>>> translateDNASequence(cds)

'MSERLSITPLGPYIGAQISGADLTRPLSDNQFEQLYHAVLRHQVVFLRDQAITPQQQ
RALAQRFGELHIHPVYPHAEGVDEIIVLDTHNDNPPDNDNWHTDVTFIETPPAGAILA
AKELPSTGGDTLWTSGIAAYEALSVPFRQLLSGLRAEHDFRKSFPEYKYRKTEEEHQR
WREAVAKNPPLLHPVVRTHPVSGKQALFVNEGFTTRIVDVSEKESEALLSFLFAHITK
PEFQVRWRWQPNDIAIWDNRVTQHYANADYLPQRRIMHRATILGDKPFYRAG*'

>>>
```

Dictionary Methods and Operations

Method or Operation	Action
d[key]	Get the value of the entry with key key in d
d[key] = val	Set the value of entry with key key to val
del d[key]	Delete entry with key key
d.clear()	Removes all entries
len(d)	Number of items
d.copy()	Makes a shallow copya
d.has_key(key)	Returns 1 if key exists, 0 otherwise
d.keys()	Gives a list of all keys
d.values()	Gives a list of all values
d.items()	Returns a list of all items as tuples (key, value)
d.update(new)	Adds all entries of dictionary new to d
d.get(key[, otherwise])	Returns value of the entry with key key if it exists Otherwise returns to otherwise
d.setdefaults(key [, val])	Same as d.get(key), but if key does not exist, sets d[key] to val
d.popitem()	Removes a random item and returns it as tuple

These materials were developed with funding from the US National Institutes of Health grant #2T36 GM008789 to the Pittsburgh Supercomputing Center

Finding ORF's

```
def findDNAORFPos(sequence, minLen, startCodon, stopCodon, startPos, endPos):
    'Finds the postion and length of the first ORF in sequence'
    while (startPos < endPos):
        startCodonPos = find(sequence, startCodon, startPos, endPos)
        if (startCodonPos >= 0):
            stopCodonPos = find(sequence, stopCodon, startCodonPos, endPos)
            if (stopCodonPos >= 0):
                if ((stopCodonPos - startCodonPos) > minLen):
                    return [startCodonPos + 3, (stopCodonPos - startCodonPos) - 3]
                else:
                    startPos = startPos + 3
            else:
                return [-1,0] # Finished the sequence without finding stop codon
        else:
            return [-1,0] # Could not find any more start codons
```

Extracting the ORF

```
def extractDNAORF(sequence, minLen, startCodon, stopCodon, startPos, endPos):
    'Returns the first ORF of length >= minLen found in sequence'
    ORFPos = findDNAORFPos(sequence, minLen, startCodon, stopCodon, startPos, endPos)
    startPosORF = ORFPos[0]
    endPosORF = startPosORF + ORFPos[1]
    if (startPosORF >= 0):
        return sequence[ORFPos[0]: ORFPos[0]+ORFPos[1]]
    else:
        return ""
```

Homework

- Design an ORF extractor to return the list of all ORF’s within a sequence together with their positions

Next Time

- Handling files containing sequences and alignments