Control flow

MBV-INFx410

Fall 2016

Indentation and scope

- Python does not use brackets or other symbols to delineate a block of code
- Python uses indentation – either tab or space
- Note: variables can only be seen and used within the block of code it is in – this is called scope

Statements

- Statements: small stand-alone pieces of code
- Simple statements:
  - print 42
  - my_list = [1, 4, 8]
  - largest = max(my_list)
- Can combine statements – often done with control flow statements

Boolean expressions – True or False

- Comparisons

A > B	A greater than B
A < B	A smaller than B
A >= B	A greater than or equal to B
A <=B	A smaller than or equal to B
A == B	A equal to B
A != B	A not equal to B

- Other values
  - True: non-empty lists, numbers != 0, and more
  - False: 0 and None

Doing comparisons

Simple comparisons

```
>>> a = 1
>>> b = 3
>>> print a == b
False
>>> print a < b
True
>>> a = "Hello"
>>> b = "World"
>>> print a != b
True
>>> print a < b
True
>>>
```

Assigning values to variables

Is 1 equal to 3?

Is 1 smaller than 3?

Assigning new values to variables

Is Hello equal to World?

Is Hello smaller than World?

Python logical operators

Boolean expression <LOGICAL OPERATOR> Boolean expression

Operator	Description	Example
AND	Logical and operator. If both left and right side are true, condition becomes true	a and b a < b and c a == b and c > d
OR	Logical or operator. If at least one of the two sides is true, condition becomes true	a or b a < b or c a == b or c > d
NOT	Logical not operator. Reverses the boolean value of a expression. If expression was true, it becomes false, and vice versa.	a and not b a < b or not c a == b and not c > d

Combining comparisons

Simple comparisons

```
>>> a = 1
>>> b = 3
>>> print a == b
False
>>> print a < b
True
>>> a = "Hello"
>>> b = "World"
>>> print a != b
True
>>> print a < b
True
>>>
```

Comparisons can be combined

```
>>> a = 1
>>> b = 4
>>> c = "Hello"
>>> d = "World"
>>> print a < b and c < d
True
>>> print a < b and c > d
False
>>> print a < b or c > d
True
>>> print a < b and not c > d
True
>>>
```

Control flow

- Control flow determines which blocks of code that will be run
- One conditional statement
  - If - elif - else
- Two iteration statements
  - For: iterate over group of elements
  - While: do until something is true

If – elif - else

- Structure:

```
if <boolean expression>:
    code block 1
elif <boolean expression>:
    code block 2
else:
    code block 3
```

Note the : (colon) has to be there!

Note: indentation is mandatory!

- Optional: elif and else
- Can have more than one elif
- Only one of these code blocks are executed
- Executed block: the one whose expression first evaluates to True (else always True)

Basic if example

- Basic test:

```
>>> fakevariable = 12
>>> if fakevariable > 10:
...     print "Var is greater than 10"
...
Var is greater than 10
>>>
```

Note indentation, done by typing Tab

Adding an else

- Can add an else to the if statement – executed as default if nothing else is true

```
>>> fakevariable = 12
>>> if fakevariable > 10:
...     print "Variable greater than 10"
... else:
...     print "Variable not greater than 10"
...
Variable greater than 10
>>>
```

First case:
If conditional true - first statement executed

```
>>> fakevariable = 9
>>> if fakevariable > 10:
...     print "Variable greater than 10"
... else:
...     print "Variable not greater than 10"
...
Variable not greater than 10
>>>
```

Second case:
If conditional not true - else statement executed

Adding elifs

- Can test on multiple conditions by introducing one or more elifs

```
>>> fakevariable = 7
>>> if fakevariable > 10:
...     print "Variable greater than 10"
... elif fakevariable > 5:
...     print "Variable greater than 5"
... else:
...     print "Variable smaller than 5"
...
Variable greater than 5
>>>
```

IF statements are order sensitive – the code inside of the first boolean expression that evaluates as True gets executed!

Testing sequence length

- Get DNA string in from command line
- Assign DNA string to variable DNA
- Calculate the length of the DNA string
- Draft on paper if statement that does the following:

If string longer than 10 nts:
prints "DNA string is longer than 10 nts"
Else if string is between 5 and 10
prints "DNA string is between 5 and 10 nts"
Otherwise, just
prints "DNA string is shorter than 5 nts"

SeqLen.py

```
import sys
# var DNA will contain the DNA string in question
DNA = sys.argv[1]
textlength = len(DNA)
if textlength > 10:
    print "DNA string is longer than 10 nts"
elif textlength > 5 and textlength < 10:
    print "DNA string is between 5 and 10 nts"
else:
    print "DNA string is shorter than 5 nts"
```

- Have script in your directory
- Run script, give it DNA strings so that all three clauses are triggered!
- Can you find out something odd about it?

For

- Structure:

for var in iterable:
    code block

Note the : (colon) has to be there!

- Code block executed for each element in iterable
- var takes on value of current element
- Iterables are:

-Strings, lists and other data types

For loop on lists

- Can use a for loop to access each element in the list:

```
>>> a = [1,2,3,4,5,6,7,8,9]
>>> for var in a:
...     print var
...
1
2
3
4
5
6
7
8
9
>>>
```

Note indentation, done by typing Tab

var is variable a is iterable

For loops and manipulation

- Can manipulate data inside the for loop:

```
>>> a = [1,2,3,4,5,6,7,8,9]
>>> for var in a:
...       print var*var
...
1
4
9
16
25
36
49
64
81
>>>
```

Print words and their lengths

- Have variable containing a list with words

  words = ["red", "green", "blue", "yellow"]
- Iterate over list and:

  print word, length of word

```
>>> words = ["red", "green", "blue", "yellow"]
>>> for word in words:
...       print word, len(word)
...
red 3
green 5
blue 4
yellow 6
>>>
```

Nested for loops

- Can have a for loop inside of another loop

```
>>> for number in [1,2,3]:
...       for letter in ["A", "B", "C"]:
...             print number, letter
...
1 A
1 B
1 C
2 A
2 B
2 C
3 A
3 B
3 C
>>>
```

```
>>> dna = "AGCT"
>>> for one in dna:
...             for two in dna:
...                   for three in dna:
...                         print one+two+three
...
AAA
AAG
AAC
AAT
AGA
AGG
AGC
....
```

- Can you print all 64 codons using five lines of code?

Combining for and if

- Often used to iterate over something, and make decisions on what to do with it
- Example: iterate over list of DNA strings- do any of them contain Ns?

OnlyDNAContent.py

```
dnaStrings = ["ATGGC", "CGNA", "TTAG", "ATC"]
for dna in dnaStrings:
    if "N" in dna:
        print dna + " is not valid"
    else:
        print dna + " does not contain Ns"
```

- File is in your directory
- Modify: can you add tests for the presence of Xes and Ys?

  Hint: elifs

While loop

- Structure

  while <boolean expression> == True:

  code block
- Important: code block MUST change truth value of expression, otherwise infinite loop

While example

- Iterate over all codons in DNA string

```
>>> DNA = "AAAGGGCCCTTTG"
>>> i = 0
>>> while i < len(DNA):
...     print DNA[i:i+3]
...     i = i + 1
...
AAA
AAG
AGG
GGG
GGC
GCC
CCC
CCT
CTT
TTG
TG
G
>>>
```

Note i = i + 1, increment the counter so that expression will at some point be false

Slicing DNA into codons

- Have DNA string in variable DNA
- Goal: print out all codons in the string
- Try modifying code on previous slide

0	1	2	3	4	5	6	7	8	9	10	11	12
A	A	A	G	G	G	C	C	C	T	T	T	G

- 1: modify i to skip to next codon

  > set i to increment with 3 instead of 1
- 2: ensure that we only get chunks of three, and not just two or one letters

  > Subtract two from length of DNA in while loop

Slicing DNA into codons

```
>>> DNA = " AAAGGGCCCTTTG"
>>> i = 0
>>> while i < len(DNA) - 2:
...       print DNA[i:i+3]
...       i = i + 3
...
AAA
GGG
CCC
TTT
>>>
```

GCdecider.py

- Copy GCcontent.py script into new file called GCdecider.py
- Change script so that,
  - If GC content is equal or above 50%, print out "GC content is equal or above 50%
  - else print out "GC content is below 50%"
- Change the DNA input to trigger the else clause

GCdecider.py

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

# count G's and C's
g = DNA.count("C")
c = DNA.count("G")
gccontent = (g+c)/float(len(DNA))*100

if gccontent >= 50.0:
    print "GC content equal or higher than 50%"
else:
    print "GC content lower than 50%"
```

TranslateProtein.py

- Goal: Translate DNA open reading frame into protein
- Need translation table: dictionary with mapping from DNA to protein
- How: while loop that iterates over string, takes three and three letters and looks that up in the translation table
- Each new amino acid added to those already translated
- Result: protein

TranslateProtein.py

```
import sys
#input on cmd line should be ATGCAGAACATA
dna = sys.argv[1]
# shortened translation table:
codon_table = {"ATA":"I", "ATG":"M", "ACA":"T", \
"AAC":"N", "CGA":"R", "CAG":"Q" }

# empty string that we can add onto
protein = ""
i = 0

# look at the while loop earlier, what would you have
# here to get triplets?
# Remember: lookup in translation table is
# codon_table[triplet], and concatenation is done
# by adding using a + sign.
```

TranslateProtein.py

```
# our string:
# dna = "ATGCAGAACATA"

# shortened translation table:
codon_table = {"ATA":"I", "ATG":"M", "ACA":"T", \
"AAC":"N", "CGA":"R", "CAG":"Q" }

# empty string that we can add onto
protein = ""
i = 0

# Iterating over the string:
while i < len(dna) - 2:
    codon = dna[i:i + 3]
    protein = protein + codon_table[codon]
    i = i + 3
print protein
```

```
[karinlag@freebee]~/teaching% python TranslateProtein.py
MQNI
[karinlag@freebee]~/teaching%
```