Strings

Genome 559: Introduction to Statistical and Computational Genomics

Prof. James H. Thomas

You run a program by typing at a terminal session command line prompt (which may be `>` or `$` or something else depending on your computer; it also may or may not have some text before the prompt).

If you type '`python`' at the prompt you will enter the Python IDLE interpreter where you can try things out (ctrl-D to exit).

If you type '`python myprog.py`' at the prompt, it will run the program '`myprog.py`' if it is present in the present working directory.

'`python myprog.py arg1 arg2`' (etc) will provide command line arguments to the program.

Each argument is a string object and they are accessed using `sys.argv[0]`, `sys.argv[1]`, etc., where the program file name is the zeroth argument.

Write your program with a text editor and be sure to save it in the present working directory before running it.

Strings

- A string type object is a sequence of characters.
- In Python, strings start and end with single or double quotes (they are equivalent but they have to match).

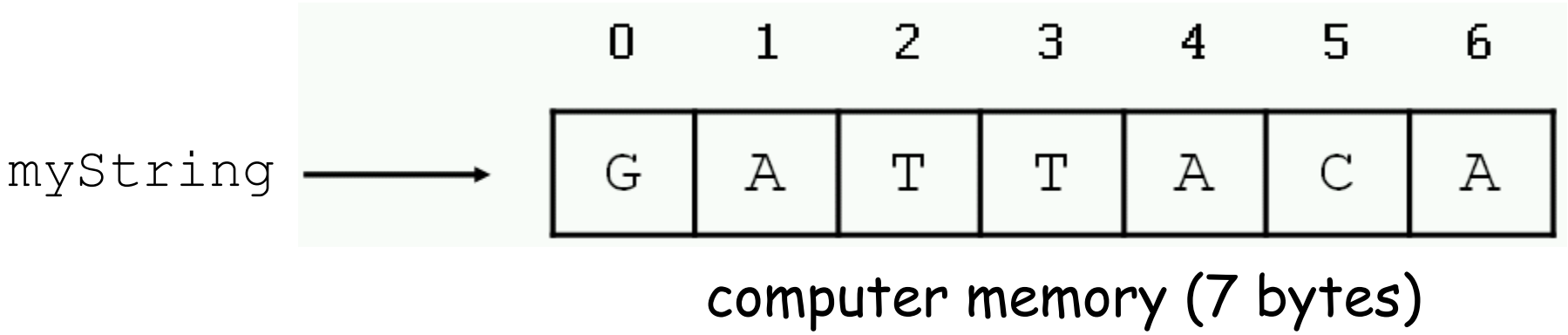
```
>>> s = "foo"
>>> print s
foo
>>> s = 'Foo'
>>> print s
Foo
>>> s = "foo'
SyntaxError: EOL while scanning string literal
```

(EOL means end-of-line; to the Python interpreter there was no closing double quote before the end of line)

Defining strings

- Each string is stored in computer memory as a list (array, vector) of characters.

```
>>> myString = "GATTACA"
```



In effect, the Python variable `myString` consists of a pointer to the position in computer memory (the address) of the 0th byte above. Every byte in your computer memory has a unique integer address.

How many bytes are needed to store the human genome? (3 billion nucleotides)

Accessing single characters

- You can access individual characters by using indices in square brackets.

```
>>> myString = "GATTACA"
>>> myString[0]
'G'
>>> myString[2]
'T'
>>> myString[-1]
'A'
>>> myString[-2]
'C'
>>> myString[7]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
```

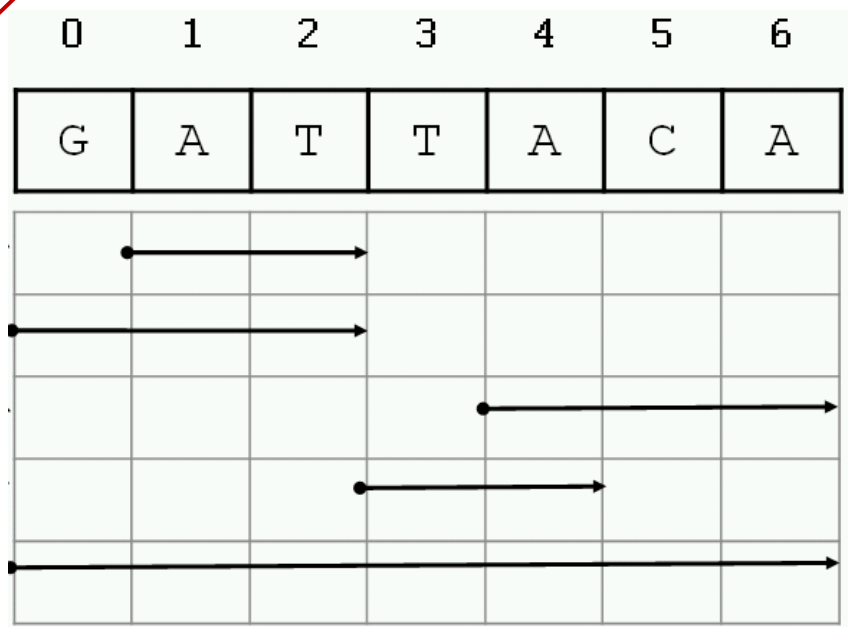
Negative indices start at the end of the string and move left.

FYI - when you request `myString[n]` Python in effect adds `n` to the address of the string and returns that byte from memory.

Accessing substrings ("slicing")

```
>>> myString = "GATTACA"
>>> myString[1:3]
'AT'
>>> myString[:3]
'GAT'
>>> myString[4:]
'ACA'
>>> myString[3:5]
'TA'
>>> myString[:]
'GATTACA'
```

shorthand for beginning or end of string



notice that the length of the returned string [x:y] is y - x

Special characters

- The backslash is used to introduce a special character.

```
>>> print "He said "Wow!""
SyntaxError: invalid syntax
>>> print "He said \"Wow!\""
He said "Wow!"
>>> print "He said:\nWow!"
He said:
Wow!
```

Escape sequence	Meaning
\\	Backslash
\'	Single quote
\"	Double quote
\n	Newline
\t	Tab

More string functionality

```
>>> len("GATTACA")
7
>>> print "GAT" + "TACA"
GATTACA
>>> print "A" * 10
AAAAAAAAAA
>>> "GAT" in "GATTACA"
True
>>> "AGT" in "GATTACA"
False
>>> temp = "GATTACA"
>>> temp2 = temp[1:4]
>>> temp2
ATT
```

←Length

←Concatenation

←Repeat

(you can read this as "is GAT in GATTACA ?")

←Substring tests

← Assign a string slice to a variable name

String methods

- In Python, a <u>method</u> is a function that is defined with respect to a particular object.
- The syntax is:

```
object.method(arguments)
```

```
>>> dna = "ACGT"
>>> dna.find("T")
3
```

the first position where "T" appears

object (in this case a string object)

string method

method argument

String methods

```
>>> s = "GATTACA"
>>> s.find("ATT")
1
>>> s.count("T")
2
>>> s.lower()
'gattaca'
>>> s.upper()
'GATTACA'
>>> s.replace("G", "U")
'UATTACA'
>>> s.replace("C", "U")
'GATTAUA'
>>> s.replace("AT", "**")
'G**TACA'
>>> s.startswith("G")
True
>>> s.startswith("g")
False
```

Function with no arguments

Function with two arguments

Strings are immutable

- Strings cannot be modified; instead, create a new string from the old one.

```
>>> s = "GATTACA"
>>> s[0] = "R"
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'str' object doesn't support item assignment
>>> s = "R" + s[1:]
>>> s
'RATTACA'
>>> s = s.replace("T","B")
>>> s
'RABBACA'
>>> s = s.replace("ACA", "I")
>>> s
'RABBI'
```

Strings are immutable

- String methods do not modify the string; they return a new string.

```
>>> seq = "ACGT"
>>> seq.replace("A", "G")
'GCGT'
>>> print seq
ACGT

>>> seq = "ACGT"
>>> new_seq = seq.replace("A", "G")
>>> print new_seq
GCGT
```

assign the result from the right to a variable name

String summary

Basic string operations:

```
S = "AATTGG"          # assignment - or use single quotes ' '
s1 + s2               # concatenate
s2 * 3                # repeat string
s2[i]                 # get character at position 'i'
s2[x:y]               # get a substring
len(S)                # get length of string
int(S)                # turn a string into an integer
float(S)              # turn a string into a floating point decimal number
```

Methods:

```
S.upper()
S.lower()
S.count(substring)
S.replace(old,new)
S.find(substring)
S.startswith(substring)
S. endswith(substring)
```

is a special character – everything after it is a comment, which the program will ignore – USE LIBERALLY!!

Printing:

```
print var1,var2,var3          # print multiple variables
print "text",var1,"text"      # print a combination of explicit text (strings) and variables
```

Tips:

Reduce coding errors - get in the habit of always being aware what type of object each of your variables refers to.

Build your program bit by bit and check that it functions at each step by running it.

Sample problem #1

- Write a program called dna2rna.py that reads a DNA sequence from the first command line argument and prints it as an RNA sequence. Make sure it retains the case of the input.

```
> python dna2rna.py  ACTCAGT
ACUCAGU
> python dna2rna.py actcagt
acucagu
> python dna2rna.py ACTCagt
ACUCagu
```

Hint: first get it working just for uppercase letters.

Two solutions

```
import sys
seq = sys.argv[1]
new_seq = seq.replace("T", "U")
newer_seq = new_seq.replace("t", "u")
print newer_seq
```

OR

```
import sys
print sys.argv[1]
```

(to be continued)

Two solutions

```
import sys
seq = sys.argv[1]
new_seq = seq.replace("T", "U")
newer_seq = new_seq.replace("t", "u")
print newer_seq
```

```
import sys
print sys.argv[1].replace("T", "U")
```

(to be continued)

Two solutions

```
import sys
seq = sys.argv[1]
new_seq = seq.replace("T", "U")
newer_seq = new_seq.replace("t", "u")
print newer_seq
```

```
import sys
print sys.argv[1].replace("T", "U").replace("t", "u")
```

- It is legal (but not always desirable) to chain together multiple methods on a single line.

Sample problem #2

- Write a program get-codons.py that reads the first command line argument as a DNA sequence and prints the first three codons, one per line, in uppercase letters.

```
> python get-codons.py TTGCAGTCG
TTG
CAG
TCG
> python get-codons.py TTGCAGTCGATCTGATC
TTG
CAG
TCG
> python get-codons.py tcgatcgactg
TCG
ATC
GAC
```

(slight challenge – print the codons on one line separated by spaces)

Solution #2

```
# program to print the first 3 codons from a DNA
# sequence given as the first command-line argument
import sys
seq = sys.argv[1]  # get first argument
up_seq = seq.upper()  # convert to upper case
print up_seq[0:3]  # print first 3 characters
print up_seq[3:6]  # print next 3
print up_seq[6:9]  # print next 3
```

These comments are simple, but when you write more complex programs good comments will make a <u>huge</u> difference in making your code understandable (both to you and others).

Sample problem #3 (optional)

- Write a program that reads a protein sequence as a command line argument and prints the location of the first cysteine residue (C).

```
> python find-cysteine.py
MNDLSGKTVIITGGARGLGAEAARQAVAAGARVVLADVLDEEGAATARELGDAARYQHLDVTI
EEDWQRVCAYAREEFGSVDGL
70
> python find-cysteine.py
MNDLSGKTVIITGGARGLGAEAARQAVAAGARVVLADVLDEEGAATARELGDAARYQHLDVTI
EEDWQRVVAYAREEFGSVDGL
-1
```

note: the `-1` here means that no C residue was found

Solution #3

```
import sys
protein = sys.argv[1]
upper_protein = protein.upper()
print upper_protein.find("C")
```

(Always be aware of upper and lower case for sequences - it is valid to write them in either case. This is handled above by converting to uppercase so that 'C' and 'c' will both match.)

Challenge problem

- Write a program get-codons2.py that reads the first command- line argument as a DNA sequence and the second argument as the frame, then prints the first three codons on one line separated by spaces.

```
> python get-codons2.py TTGCAGTCGAG 0
TTG CAG TCG
> python get-codons2.py TTGCAGTCGAG 1
TGC AGT CGA
> python get-codons2.py TTGCAGTCGAG 2
GCA GTC GAG
```

Challenge solution

```
import sys
seq = sys.argv[1]
frame = int(sys.argv[2])
seq = seq.upper()
c1 = seq[frame:frame+3]
c2 = seq[frame+3:frame+6]
c2 = seq[frame+6:frame+9]
print c1, c2, c3
```

Reading

- Chapters 2 and 8 of *Think Python* by Downey.