

Strings

Basic string operations:

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

Methods:

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

Printing:

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

Lists

A list is an ordered collection of objects.

Lists are

- ordered left to right
- indexed like strings (from 0)
- mutable
- heterogeneous

Lists and strings are similar, but lists can be changed whereas strings are immutable.

Basic list operations:

```
L = ['dna','rna','protein'] # list assignment
L2 = [1,2,'dogma',L]        # list hold different objects
L2[2] = 'central'           # change an element (mutable)
L2[0:2] = 'ACGT'            # replace a slice
del L[0:1] = 'nucs'         # delete a slice
L2 + L                       # concatenate
L2*3                         # repeat list
L[x:y]                       # define the range of a list
len(L)                       # length of list
".join(L)"                   # convert a list to string
S.split(x)                   # convert string to list- x delimited
list(S)                      # convert string to list - explode
list(T)                      # converts a tuple to list
```

Methods:

| | |
|---------------|--------------------------------------|
| L.append(x) | # add to the end |
| L.extend(x) | # append each element from x to list |
| L.count(x) | # count the occurrences of x |
| L.index(x) | # give element location of x |
| L.insert(i,x) | # insert at element x at element i |
| L.remove(x) | # delete first occurrence of x |
| L.pop(i) | # extract element I |
| L.reverse() | # reverse list in place |
| L.sort() | # sort list in place |

Numbers

Several types of numbers are defined in Python:

- Integers (1234)
- Floating point numbers (12.34)
- Octal and hexadecimal numbers (0177, 0x9gff)
- Complex numbers (3.0+4.1j)

The float() and int() functions can be used to convert a string to a number. The '%' operator converts a number to a string:

Conversion Codes:

- %d = integer
- %f = decimal value
- %e = scientific notation
- %g = easily readable notation (i.e., use decimal notation unless there are too many zeroes, then switch to scientific notation)

```
>>> "%f" % 3
'3.000000'
>>> "%.2f" % 3
'3.00'
```

File input and output

The open() command returns a file object.

```
<filehandle> = open(<filename>, <access type>)
```

Python can read, write or append to a file:

- 'r' = read
- 'w' = write
- 'a' = append

The readlines() command puts all the lines of a file into a list of strings.

The readline() command returns the next line of the file as a string.

You can write to a file using the `<file>.write()` method. The method takes a string as an input, and does not automatically append an end-of-line character. To write to a file, the file must be open with the “w” option.

If statements

A block is a group of statements that belong together. In Python, blocks are indicated by consistent indentation. An if statement consist of a logical test, followed be a ‘:’ and a block of statements that are executed if the test is true.

Here are some comparison operators to use in logical tests:

Boolean: and, or, not

Numeric: <, >, ==, !=, <>, >=, <=

String/list/dictionary: in

Note: = assigns a value to a variable name, == is a logical test for equality

Loops

The for loop allows you to perform an operation for each element in a list. As with if statements, a multiline block consisting of multiple statements can be used. Each statement must have the same indentation. It is often useful to use an integer variable to keep track of the index during a loop.

The `range()` function is also useful. The `range()` function returns a list of integers covering a specified range:

```
range([start,] stop [,step])
```

```
>>> range(5), range(2,5), range(0,10,2)
([0, 1, 2, 3, 4], [2, 3, 4], [0, 2, 4, 6, 8])
```

The `break`, `continue`, and `loop else` statements are used to control the flow of your program in loops.

break: jumps out of the closest enclosing loop

continue: jumps to the top of the closet enclosing loop

Loop else: runs if and only if the loop is exited normally (without using a `break` statement)

Loops and if statements can be nested inside of each other. Each nested block must have consistent indentation.

Dictionaries

Rules for dictionaries:

The first item is a “key”

Each key can only appear once

A key must be an immutable object: number, string or tuple

Lists cannot be keys (they are mutable)

The key should be the item you'll use to do look-ups.

Dictionary Summary

Initialize an empty dictionary

```
<dictionary> = {}
```

Store an entry:

```
<dictionary>[<key>] = <value>
```

Retrieve an entry

```
<dictionary>[<key>]
```

Check whether a dictionary has a particular key:

```
<dictionary>.haskey(<key>)
```

Retrieve an unordered list of keys or values:

```
<dictionary>.keys()
```

```
<dictionary>.values()
```