Lecture 13
Python: containers, loops, script arguments

Course: Practical Bioinformatics (BIOL 4220)
Instructor: Michael Landis
Email: michael.landis@wustl.edu

Lecture 13 outline

Last time: variables, operators, if-statements, functions

This time: Python (2 of 4)

Python

- containers
- loops
- script arguments

Containers

Containers are variables that store multiple values of the same type, called ***elements***

Container elements are generally accessed through the ***index*** operator, *[idx]*

```
>>> # create list called `x`
>>> x = [ 10, 20, 30 ]
>>> # what is the value of `x`?
>>> x
[10, 20, 30]
>>> # access the index-0 element
>>> x[0]
10
```

Lists

List elements are indexed by integers;
lists can be modified after creation (*mutable*)

```
>>> x = [ 10, 20, 30 ] # create list called `x`
>>> x # what is the value of `x`?
[10, 20, 30]
>>> x[0] = 11 # set value of index-0 element
>>> x[1:3] = [ 22, 33 ] # set values of index-1,2 elements
>>> x[3] = 55 # access the index-3 element
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> x.append(55) # append element to end of list
>>> x.insert(3, 44) # insert value 44 after 3rd index
>>> x
[11, 22, 33, 44, 55]
```

Tuples

Tuples are integer-indexed containers, but unlike lists, their contents cannot be modified (*immutable*)

```
>>> x = ( 10, 20, 30 ) # create tuple called `x`
>>> x                  # what is the value of `x`?
(10, 20, 30)
>>> x[0]
11
>>> x[0] = 11          # set value of index-0 element
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Dictionaries

Dictionaries contain *key-value* pairs;
keys are used to index values

```
>>> x = { 'a':1, 'b':2 }  # create dictionary with two key-values
>>> x                     # report value of dictionary
{'a': 1, 'b': 2}
>>> x['a’]                # retrieve dictionary value with key 'a'
1
>>> x['c'] = 3            # assign value 3 to key 'c'
>>> x.keys()              # print container of sorted keys
dict_keys(['a', 'b', 'c'])
>>> x.values()            # print container of sorted values
dict_values([1, 2, 3])
>>> x.values()[0]         # dict_values can't be accessed by index!?
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'dict_values' object does not support indexing
>>> list(x.values())[0]   # typecast dict_values as list
1
```

Alternatives for initializing lists and dicts

```
>>> x = [] # creates an empty list
>>> x
[]
>>> x = [0]*10 # creates size-10 list with values 0
>>> x
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> x = {} # creates empty dict
>>> x
{}
>>> x = dict.fromkeys(['a','b'], 0) # creates keys with values 0
{'a': 0, 'b': 0}
```

Alternatives for initializing lists and dicts

```
>>> x = [ 1, 0.1 ]
>>> x
[1, 0.1]
>>> type(x)
<class 'list'>
>>> type(x[0])
<class 'int'>
>>> type(x[1])
<class 'float'>
>>> x = [ 'a', [ 1, 0.1, {'a':0, 'b':[0]*2} ], False ]
>>> x
['a', [1, 0.1, {'a': 0, 'b': [0, 0]}], False]
```

for-loops over list elements

Executes code block while iterating
over each element in a container

```
# create list
x = [ 'a', 'b', 'c', 'd', 'e' ]
# get list length
n = len(x)
# loop over each element in list
for i in x:
    # code block
    s = i + ' (out of ' + str(n) + ')'
    print(s)

# done
print('...done!')
```

code

```
a (out of 5)
b (out of 5)
c (out of 5)
d (out of 5)
e (out of 5)
...done!
```

output

for-loops over indices

The *range(n)* function creates a list of integers with values *[0, 1, ..., n-1]*

```
# create list
x = [ 'a', 'b', 'c', 'd', 'e' ]
# loop over each integer in range
for i in range(len(x)):
    # code block
    s = x[i] + ' (out of ' + str(len(x)) + ')'
    print(s)

# done
print('...done!')
```

code

```
a (out of 5)
b (out of 5)
c (out of 5)
d (out of 5)
e (out of 5)
...done!
```

output

for-loops over dictionary items

Iterate over *(key, value)* ***items*** in a dictionary

```
# create dictionary
x = {'a':1, 'b':2, 'c':3}
# loop over all items in dictionary, while
# storing key and value for each item
for key,value in x.items():
    # code block
    s = 'key = ' + str(key) + '; '
    s += 'value = ‘ + str(value)
    print(s)

# done
print('...done!')
```

code

```
key = a; value = 1
key = b; value = 2
key = c; value = 3
...done!
```

output

enumerate

The *enumerate(x)* function assigned pairs an index to each iterable element in the container: *(index, value)*

```
# create dictionary
x = [10, 20, 30]
# create loop
for i,v in enumerate(x):
    # code block
    s = 'iteration = ' + str(i) + '; '
    s += 'value = ' + str(v)
    print(s)

# done
print('...done!')
```

code

```
iteration = 0; value = 10
iteration = 1; value = 20
iteration = 2; value = 30
...done!
```

output

Nested containers and loops

```
# create array of input
x = [[ 1,  4,  9],
     [16, 25, 36],
     [49, 64, 81]]

# create empty array for output
y = []

# iterate over rows
for i,row in enumerate(x):
  # create empty row for results
  y.append([])
  # iterate over column-values
  for j,val in enumerate(row):
    # get square root of input value
    y_ij = int( val**(1/2) )
    # store result in y[i][j]
    y[i].append(y_ij)

# print square roots
print(y)
```

code

Contains may be *nested* as elements within larger containers; for-loops may also be *nested* to process all containers, subcontainers, etc.

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

output

System arguments

Python programs can accept user arguments through the system argument vector, *sys.argv*

```
# load system library
import sys

# print each argument
print('sys.argv contents:')
for i,v in enumerate(sys.argv):
    print(' ' + str(i) + ' : ' + str(v))

# done
print('...done!')
```

Pass arguments to Python when calling the script from shell

```
$ python example.py 10 20
sys.argv contents:
0 : ./example.py
1 : 10
2 : 20
...done!
```

Overview for Lab 13