Introduction to Python 2

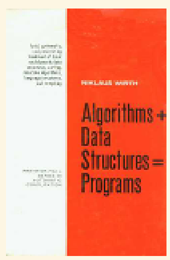
Chang Y. Chung

Office of Population Research

01/14/2014

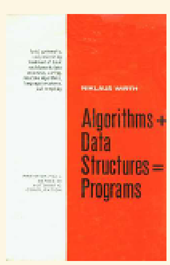
Algorithms + Data Structures = Programs

- Niklaus Wirth (1976)[3]



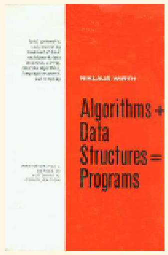
Algorithms + Data Structures = Programs

- Niklaus Wirth (1976)[3]
- Python's built-in data structures include:
  - Lists
  - Dictionaries
  - Tuples



Algorithms + Data Structures = Programs

- Niklaus Wirth (1976)[3]
- Python’s built-in data structures include:
  - Lists
  - Dictionaries
  - Tuples
- We will also briefly talk about:
  - Classes
  - Exception Handling



List

- Ordered (indexed) collection of arbitrary objects.
- Mutable – may be changed in place.

List

- Ordered collection of arbitrary objects.

```
L = []                      # a new empty list
L = list()                  # ditto

L = [1, 2.5, "abc", [56.7, 78.9]]
print len(L)                # 4
print L[1]                  # 2.5 (zero-based)
print L[3][0]               # 56.7

for x in L:
    print x
# 1
# 2.5
# "abc"
# [56.7, 78.9]

print "abc" in L, L.count("abc"), L.index("abc")
# True 1 2
```

List

- Mutable – may be changed in place.

```
L = []
L.append(5)
print L              # [5]

L[0] = 23
print L              # [23]

M = [87, 999]
L.extend(M)          # or L += M
print L              # [23, 87, 999]

del L[2]
print L              # [23, 87]
```


List

- More examples.

```
def squares(a_list):
    s = []
    for el in a_list:
        s.append(el ** 2)
    return s

sq = squares([1,2,3,4])
print sq, sum(sq)
# [1, 4, 9, 16] 30
```


List

- More examples.

```
def squares(a_list):
    s = []
    for el in a_list:
        s.append(el ** 2)
    return s

sq = squares([1,2,3,4])
print sq, sum(sq)
# [1, 4, 9, 16] 30
```


- Aliasing vs copying

```
L = [1,2,3,4]
M = L            # aliasing
L[0] = 87
print M          # [87, 2, 3, 4]

L = [1,2,3,4]
M = list(L)      # (shallow) copying. M = L[:] also works
L[0] = 87
print M          # [1,2,3,4]
```

Quiz

- Given a list,

```python
L = [1, 2, [3, 4], 5, "xyz"]
```

evaluate the following expressions:

```python
L[1] == 1
len(L) == 5
L[2] == 3, 4

[3] in L
L.index("xyz") == 4
L[-1] == "xyz"
L[-1][-1] == "z"

any([1, 2, 3]) == True
L[9] == None
len([0,1,2,]) == 3
```


Quiz

- Write a function that, given a list of integers, returns a *new* list of odd numbers only. For instance, given the list, [0,1,2,3,4], this function should return a new list, [1,3]. (Hint: Create a new empty list. Loop over the old one appending only odd numbers into the new one. Return the new one.)

Quiz

- Write a function that, given a list of integers, returns a *new* list of odd numbers only. For instance, given the list, [0,1,2,3,4], this function should return a new list, [1,3]. (Hint: Create a new empty list. Loop over the old one appending only odd numbers into the new one. Return the new one.)
- An answer.

```python
def only_odd(a_list):
    L = []
    for el in a_list:
        if el % 2 == 1:
            L.append(el)
    return L

print only_odd([0, 1, 2, 3, 4])
# [1, 3]
```

Quiz (cont.)

- (tricky) Write a function similar to the previous one. This time, however, do not return a new list. Just modify the given list so that it has only the odd numbers.
  (Hint: `del L[0]` removes the first element of the list, `L`)

Slice index

- Applies to any sequence types, including `list`, `str`, `tuple`, ...
- Has three (optional) parts separated by a colon (:), `start : end : step`, indicating `start` through but not past `end`, by `step`; Indices point *in-between* the elements.

```
    +---+---+---+---+---+---+
    | p | y | t | h | o | n |
    +---+---+---+---+---+---+
    0   1   2   3   4   5   6
   -6  -5  -4  -3  -2  -1
```

- Examples:

```python
L = ["p", "y", "t", "h", "o", "n"]
print L[:2]        # ["p", "y"] first two
print L[1:3]       # ["y", "t"]
print L[0:5:2]     # ["p", "t", "o"]
print L[-1]        # n the last element
print L[:]         # ["p", "y", "t", "h", "o", "n"] a (shallow) copy
print L[3:]        # ["h", "o", "n"]
print L[-2:]       # ["o", "n"] last two
print L[::-1]      # ["n", "o", "h", "t", "y", "p"] reversed
```


Quiz

- Suppose that you collect friendship network data among six children, each of whom we identify with a number: 0, 1, . . . , 5. The data are represented as a list of lists, where each element list represents the element child's friends.

```python
L = [[1, 2], [0, 2, 3], [0, 1], [1, 4, 5], [3, 5], [3]]
```

  For instance, the kid 0 friends with the kids 1 and 2, since `L[0] == [1, 2]` Calculate the average number of friends the children have. (Hint: `len()` returns the list size.)

Quiz

- Suppose that you collect friendship network data among six children, each of whom we identify with a number: 0, 1, . . . , 5. The data are represented as a list of lists, where each element list represents the element child's friends.

```
L = [[1, 2], [0, 2, 3], [0, 1], [1, 4, 5], [3, 5], [3]]
```

  For instance, the kid 0 friends with the kids 1 and 2, since `L[0] == [1, 2]` Calculate the average number of friends the children have. (Hint: `len()` returns the list size.)

- An answer:

```
total = 0.0          # make total a float type
for el in L:
    total += len(el)
avg = total / len(L)
print avg
# 2.1666
```


Quiz (cont.)

- (tricky)Write a function to check if *all* the friendship choices are reciprocated. It should take a list like previous one and return either `True` or `False`. (Hint: You may want to use a utility function below.)

```
def mutual(a_list, ego, alter):
    return alter in a_list[ego] and ego in a_list[alter]
```


List Comprehension

- A concise way to create a list. An example:

```
[x for x in range(5) if x % 2 == 1]        # [1, 3]
```

- An equivalent code using the for loop:

```
L = []
for x in range(5):
    if x % 2 == 1:
        L.append(x)                        # [1, 3]
```

- More examples.

```
[x - 5 for x in range(6)]                  # [-5, -4, -3, -2, -1, 0]
[abs(x) for x in [-2,-1,0,1]]              # [2, 1, 0, 1]
[x for x in range(6) if x == x**2]         # [0, 1]
[1 for x in [87, 999, "xyz"]]              # [1, 1, 1]
[x - y for x in range(2) for y in [7, 8]]  # [-7, -8, -6, -7]
```

Dictionary

- A collection of key-value pairs.
- Indexed by keys.
- Mutable.

Dictionary

- A collection of key-value pairs.
- Indexed by keys.
- Mutable.
- Also known as associative array, map, symbol table, . . .
- Usually implemented as a hash table.

Dictionary

- A collection of key-value pairs, indexed by keys.

```
D = {}                          # an empty dictionary. D=dict() also works

D["one"] = 1                    # {"one": 1}
D["two"] = 2
print D                         # {"one": 1, "two": 2}

print D.keys()                  # ["two", "one"] arbitrary order!
print "three" in D.keys()       # False. "three" in D also works

D = {"Apple": 116, "Big Mac": 550}

for key in ["Apple", "Orange", "Big Mac"]:
    if key in D:
        value = D[key]
        print "{0} has {1} calories".format(key, value)
    else:
        print "{0} is not found in the dictionary".format(key)
# Apple has 116 calories
# Orange is not found in the dictionary
# Big Mac has 550 calories
```

Dictionary

- More Dictionary examples.

```
D = {"China": 1350, "India":1221, "US":317}
for key in D.keys():
    print "Pop of {0}: {1} mil".format(key, D[key])
# Pop of India: 1221 mil
# Pop of China: 1350 mil
# Pop of US: 317 mil

D = {[1,2]: 23}
# TypeError: unhashable type: 'list'

D = {2: [2, 3], 200: [3, 4], 95: [4, 5]} # OK
print D[2]         # [2, 3]
print D[200]       # [3, 4]
```


A Data Structure

- SAT has three subsections: Critical Reading, Mathematics, and Writing. A result of taking an SAT exam is three scores.

```
# data
SAT = {"cr":780, "m":790, "w":760}
# usage
print SAT["m"]              # 790
```


A Data Structure

- SAT has three subsections: Critical Reading, Mathematics, and Writing. A result of taking an SAT exam is three scores.

```
# data
SAT = {"cr":780, "m":790, "w":760}
# usage
print SAT["m"]              # 790
```


- You can take SAT exams more than once.

```
# data
SATs = [{"cr":780, "m":790, "w":760},
        {"cr":800, "m":740, "w":790}]
# usage
print SATs[0]               # {"cr":780, "m":790, "w":760}
print SATs[0]["cr"]         # 780
```

More Complicated Data Structure

- Hypothetical SAT data for two people: Jane and Mary.

```
SAT = {"Jane": {"lastname": "Thompson",
                 "test": [{"cr": 700, "m": 690, "w":710}] },
       "Mary": {"lastname": "Smith",
                 "test": [{"cr": 780, "m": 790, "w":760},
                          {"cr": 800, "m": 740, "w":790}] }}

print SAT["Jane"]
# {"test": ["cr": 700, "m": 690, "w": 710], "lastname": "Thompson"}

print SAT["Jane"]["lastname"]         # Thompson
print SAT["Jane"]["test"]             # [{"cr": 700, "m": 690, "w":710}]
print SAT["Jane"]["test"][0]          # {"cr": 700, "m": 690, "w": 710}
print SAT["Jane"]["test"][0]["cr"]    # 700

mary1 = SAT["Mary"]["test"][1]
print mary1["cr"]                     # 800
```


Quiz

- Make a dictionary of 2012 SAT percentile ranks for the scores from 660 to 700 and for all three subsections. The full table is available at `http://tinyurl.com/k38xve8`. Given this dictionary, say `D`, a lookup, `D[660]["cr"]` should be evaluated to `91`.

Quiz

- Make a dictionary of 2012 SAT percentile ranks for the scores from 660 to 700 and for all three subsections. The full table is available at `http://tinyurl.com/k38xve8`. Given this dictionary, say `D`, a lookup, `D[660]["cr"]` should be evaluated to `91`.
- An answer.

```
D = {700: {"cr": 95, "m": 93, "w": 96},
     690: {"cr": 94, "m": 92, "w": 95},
     680: {"cr": 93, "m": 90, "w": 94},
     670: {"cr": 92, "m": 89, "w": 93},
     660: {"cr": 91, "m": 87, "w": 92}}

print D[660]["cr"]       # 91
```

Quiz (cont.)

- (tricky) Write a new dictionary `DD` such that we look up the subsection first and then the score. That is, `DD["cr"][660]` should be evaluated to `91`.
  (Hint: Start with a dictionary below.):

```
DD = {"cr": {}, "m": {}, "w": {}}
```


Tuples

- A sequence of values separated by commas.
- Immutable.
- Often automatically *unpacked*.

Tuples

- A sequence of values separated by commas. Immutable.

```
T = tuple()                  # empty tuple. T = () works also
N = (1)                      # not a tuple
T = (1, 2, "abc")            # a tuple (1, 2, "abc")
print T[0]                   # 1
T[0] = 9                     # TypeError. immutable
```

- Often automatically unpacked.

```
T = (2, 3)
a, b = T                     # a is 2, b is 3
a, b = b, a                  # a and b swapped.

D = {"x": 23, "y": 46}
D.items()                    # [("y", 46), ("x", 23)]
for k, v in D.items():
    print "%s ==> %d" % (k, v)   # y ==> 46
                                 # x ==> 23
```

Class

- class defines a (user-defined) type, a grouping of some data (properties) and functions that work on the data (methods).
- An object is an *instance* of a type.
- Examples:
  - int is a type; 23 is an object.
  - str a type; "abc" an object.
  - "word document file" a type; "my_diary.docx" is an object
  - We have been using objects.

Examples of Built-in Types

- The str type has a bunch of methods.

```
"abc".upper()                          # ABC
"abc".find("c")                        # 2
"abc".split("b")                       # ["a", "c"]
```

- open() function returns a file object (representing an opened file).

```
with open("test.txt", "w") as my_file:
    my_file.write("first line\n")
    my_file.write("second line\n")
    my_file.write("third line")

    print type(my_file)                # <type "file">
    print dir(my_file)                 # properties and methods

my_file.write("something")             # error. I/O on closed file
```


Class

- Let’s create a bank account type.

```
class BankAccount:

    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount
```

- Usage examples.

```
my_account = BankAccount(100)
my_account.withdraw(5)
print my_account.balance               # 95

your_account = BankAccount()
your_account.deposit(100)
your_account.deposit(10)
print your_account.balance             # 110
```

Quiz

- Implement a Person type(or class) which has three properties (first_name, last_name, and birth_year); and two methods: full_name() and age(). The age() method should take the current year as an argument. You may use the template below.

```
class Person:
    def __init__(self, first, last, year):
        pass
    def full_name(self):
        pass
    def age(self, current_year):
        pass

# check
mr_park = Person("Jae-sang", "Park", 1977)
print mr_park.full_name()              # Jae-sang Park
print mr_park.age(2014)                # 37
```


Inheritance

- A mechanism for code reuse in object-oriented programming (OOP).
- A subtype is a specialized basetype.

```
import webbrowser

class CoolPerson(Person):
    def __init__(self, name, birth_year, video):
        Person.__init__(self, name, None, birth_year)
        self.video = video
    def full_name(self):
        return self.first_name
    def show_off(self):
        url = "http://www.youtube.com/watch?v={0}"
        webbrowser.open(url.format(self.video))

# check
psy = CoolPerson("PSY", 1977, "9bZkp7q19f0")
print psy.full_name()              # PSY
print psy.age(2012)                # 35
psy.show_off()                     # show off the style
```


Exception Handling

- An exception is raised when a (run-time) error occurs. By default, the script stops running immediately.

```
L = [0, 1, 2, 3]
print L[5]
# IndexError: list index out of range
```

- `try: ... except: ...` let us catch the exception and handle it.

```
L = [0, 1, 2, 3]
try:
    print L[5]

except IndexError:
    print "no such element"

print "next"
# no such element
# next
```

Throwing Exception

- We can raise (or throw) an exception as well.

```
def fetch(a_list, index):
    if index >= len(a_list):
        raise IndexError("Uh, oh!")
    return a_list[index]

print fetch(L, 5)
# IndexError: Uh, oh!
```

- Script can keep going if you catch and handle the exception.

```
L = [0, 1, 2, 3]
try:
    print fetch(L, 5)  # this raises an exception
except IndexError:
    print "an exception occurred"
print "next"
# an exception occurred
# next
```


An Example

- urlopen() in urllib2 module raises an exception when the web page is not found.

```
import urllib2

L = ["http://google.com",
     "http://google.com/somethingfantastic",
     "http://yahoo.com"]

# we want to open each page in turn
for url in L:
    try:
        page = urllib2.urlopen(url)
        print page.getcode()
    except urllib2.HTTPError:
        print "failed to open: {0}".format(url)

# 200 (a return code of 200 means OK)
# failed to open: http://google.com/somethingfantastic
# 200
```


A Data Structure Usage Example

- STAN (`http://mc-stan.org`) is a C++ library / language implementing Markov chain Monte Carlo sampling (NUTS, HMC).
- STAN provides three application programming interfaces (or API's): R, Python, and shell
- This is an example of using the Python API, which is provided in a Python module, PyStan[1].
- In order to run this, you need to install: Cython (`http://cython.org`), NumPy (`http://www.numpy.org`), and STAN itself.
- From PyStan doc (`http://tinyurl.com/olap8sx`), fitting the eight school model in Gelman et al. [2, sec 5.5].

Data Structure Usage Example (cont.)

- Import PyStan module and put STAN code in a string.

```
import pystan
schools_code = """
data {
    int<lower=0> J; // number of schools
    real y[J]; // estimated treatment effects
    real<lower=0> sigma[J]; // s.e. of effect estimates
}
parameters {
    real mu;
    real<lower=0> tau;
    real eta[J];
}
transformed parameters {
    real theta[J];
    for (j in 1:J)
        theta[j] <- mu + tau * eta[j];
}
model {
    eta ~ normal(0, 1);
    y ~ normal(theta, sigma);
}
"""
```


Data Structure Usage Example (cont.)

- cont.

```
schools_data = {"J": 8,
                "y": [28,  8, -3,  7, -1,  1, 18, 12],
                "sigma": [15, 10, 16, 11,  9, 11, 10, 18]}

fit = pystan.stan(model_code=schools_code,
                  data=schools_data, iter=1000, chains=4)

la = fit.extract(permuted=True)
mu = la["mu"]
# do something with mu here

print str(fit)    # (nicely) print fit object
fit.plot()        # requires matplotlib
```

- Notice that:
  - Input data are supplied in a dictionary.
  - `stan()` function in the module runs the model.
  - The function returns a `fit` type object, which has several methods including `extract()` and `plot()`.

Data Structure Usage Example (cont.)

- Output, in part

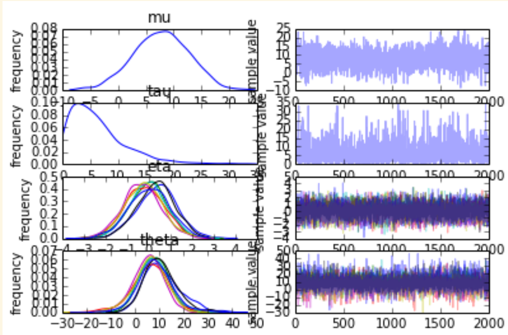
```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model NOW.
Inference for Stan model: anon_model.
4 chains, each with iter=1000; warmup=500; thin=1;
post−warmup draws per chain=500, total post−warmup draws=2...

          mean se_mean    sd  2.5%   25%   50%   75% 97.5% n_eff...
mu         7.8     0.2   5.1  −2.0   4.4   7.9  11.3  17.2 515.0...
tau        6.4     0.3   5.4   0.4   2.6   5.1   8.6  20.5 362.0
eta[0]     0.4     0.0   0.9  −1.5  −0.2   0.4   1.0   2.2 597.0
eta[1]    −0.0     0.0   0.9  −1.8  −0.6  −0.0   0.5   1.7 582.0
...
theta[6]  10.4     0.3   6.9  −1.9   5.7   9.8  14.3  25.8 594.0
theta[7]   8.3     0.3   7.5  −6.2   3.7   8.0  12.7  25.0 604.0
lp__      −4.9     0.1   2.6 −10.5  −6.5  −4.7  −3.2  −0.3 318.0

Samples were drawn using NUTS(diag_e) at Thu Jan  9 17:53:
For each parameter, n_eff is a crude measure of effective
and Rhat is the potential scale reduction factor on split
convergence, Rhat=1).
```

Data Structure Usage Example (cont.)

- Plots



Summary

- List – An ordered collection of objects. Mutable.
- Dictionary – A collection of key-value pairs. Mutable.
- Tuple – A sequence of values separated by commas. Immutable.
- Class – Defines a type, a grouping of properties and methods.
- try: ... except: ... – Catch and handle exceptions.

References

Stan project team site.
`http://mc-stan.org/team.html`.

Andrew Gelman, John B. Carlin, H. S. S. D. B. R.
Bayesian Data Analysis, 2nd ed.
Chapman & Hall/CRC Texts in Statistical Science. Chapman and Hall/CRC, July 2003.

Wirth, N.
Algorithms + Data Structures = Programs, 1st ed.
Prentice Hall Series in Automatic Computation. Prentice-Hall, February 1976.