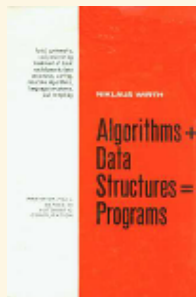


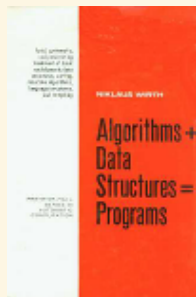
# Algorithms + Data Structures = Programs

- ▶ Niklaus Wirth (1976)[?]



# Algorithms + Data Structures = Programs

- ▶ Niklaus Wirth (1976)[?]
- ▶ Python's built-in data structures include:
  - ▷ List
  - ▷ Dictionary
  - ▷ Tuple



# Algorithms + Data Structures = Programs

- ▶ Niklaus Wirth (1976)[?]
- ▶ Python's built-in data structures include:
  - ▷ List
  - ▷ Dictionary
  - ▷ Tuple
- ▶ We will also briefly talk about:
  - ▷ Class
  - ▷ Exception Handling



# List

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

# List

- ▶ Ordered collection of arbitrary objects.

```
1 L = []                # a new empty list
2 L = list()           # ditto
3
4 L = [1, 2.5, "abc", [56.7, 78.9]]
5 print len(L)         # 4
6 print L[1]           # 2.5 (zero-based)
7 print L[3][0]        # 56.7
8
9 for x in L:
10     print x
11 # 1
12 # 2.5
13 # "abc"
14 # [56.7, 78.9]
15
16 print "abc" in L, L.count("abc"), L.index("abc")
17 # True 1 2
```

# List

- ▶ Mutable – may be changed in place.

```
1 L = []
2 L.append(5)
3 print L           #[5]
4
5 L[0] = 23
6 print L           #[23]
7
8 M = [87, 999]
9 L.extend(M)       # or L += M
10 print L          #[23, 87, 999]
11
12 del L[2]
13 print L          #[23, 87]
```

# List

## ► More examples.

```
1 def squares(a_list):
2     s = []
3     for el in a_list:
4         s.append(el ** 2)
5     return s
6
7 sq = squares([1,2,3,4])
8 print sq, sum(sq)
9 #[1, 4, 9, 16] 30
```

# List

## ► More examples.

```
1 def squares(a_list):
2     s = []
3     for el in a_list:
4         s.append(el ** 2)
5     return s
6
7 sq = squares([1,2,3,4])
8 print sq, sum(sq)
9 #[1, 4, 9, 16] 30
```

## ► Aliasing vs copying

```
1 L = [1,2,3,4]
2 M = L           # aliasing
3 L[0] = 87
4 print M        #[87, 2, 3, 4]
5
6 L = [1,2,3,4]
7 M = list(L)    #(shallow) copying. M = L[:] also works
8 L[0] = 87
9 print M        #[1,2,3,4]
```



# Quiz

► Given a list,

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

evaluate the following expressions:

```
1 L[1] == 1
2 len(L) == 5
3 L[2] == 3, 4
4
5 [3] in L
6 L.index("xyz") == 4
7 L[-1] == "xyz"
8 L[-1][-1] == "z"
9
10 any([1, 2, 3]) == True
11 L[9] == None
12 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.

```
1 def only_odd(a_list):
2     L = []
3     for el in a_list:
4         if el % 2 == 1:
5             L.append(el)
6     return L
7
8 print only_odd([0, 1, 2, 3, 4])
9 #[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`)

```
1 def odd_only(a_list):
2     i = 0
3     length = len(a_list)
4     while i < length:
5         if (n % 2) == 0:
6             del a_list[i]
7             i -= 1
8             length -= 1
9         i += 1
10 L = [0, 1, 2, 3, 4] # check
11 odd_only(L)
12 print 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.

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

## ▶ Examples:

```
1 L = ["p", "y", "t", "h", "o", "n"]
2 print L[:2]           # ["p", "y"] first two
3 print L[1:3]         # ["y", "t"]
4 print L[0:5:2]       # ["p", "t", "o"]
5 print L[-1]          # n the last element
6 print L[:]           # ["p", "y", "t", "h", "o", "n"] a (shallow) copy
7 print L[3:]          # ["h", "o", "n"]
8 print L[-2:]         # ["o", "n"] last two
9 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.

```
1 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.

```
1 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:

```
1 total = 0.0                # make total a float type
2 for el in L:
3     total += len(el)
4 avg = total / len(L)
5 print avg
6 # 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.)

```
1 def mutual(a_list, ego, alter):
2     return alter in a_list[ego] and ego in a_list[alter]
3
4 def all_reciprocated(a_list):
5     for ego in range(len(a_list)):
6         alters = a_list[ego]
7         for alter in alters:
8             if not mutual(a_list, ego, alter):
9                 return False
10    return True
```



# List Comprehension

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

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

- ▶ An equivalent code using the for loop:

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

- ▶ More examples.

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

```
1 D = {} # an empty dictionary. D=dict() also works
2
3 D["one"] = 1 # {"one": 1}
4 D["two"] = 2
5 print D # {"one": 1, "two": 2}
6
7 print D.keys() # ["two", "one"] arbitrary order!
8 print "three" in D.keys() # False. "three" in D also works
9
10 D = {"Apple": 116, "Big Mac": 550}
11
12 for key in ["Apple", "Orange", "Big Mac"]:
13     if key in D:
14         value = D[key]
15         print "{0} has {1} calories".format(key, value)
16     else:
17         print "{0} is not found in the dictionary".format(key)
18 # Apple has 116 calories
19 # Orange is not found in the dictionary
20 # Big Mac has 550 calories
```

# Dictionary

## ► More Dictionary examples.

```
1 D = {"China": 1350, "India":1221, "US":317}
2 for key in D.keys():
3     print "Pop of {0}: {1} mil".format(key, D[key])
4 # Pop of India: 1221 mil
5 # Pop of China: 1350 mil
6 # Pop of US: 317 mil
7
8 D = {[1,2]: 23}
9 # TypeError: unhashable type: 'list'
10
11 D = {2: [2, 3], 200: [3, 4], 95: [4, 5]} # OK
12 print D[2]      # [2, 3]
13 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.

```
1 # data
2 SAT = {"cr":780, "m":790, "w":760}
3 # usage
4 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.

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

- ▶ You can take SAT exams more than once.

```
1 # data
2 SATs = [{"cr":780, "m":790, "w":760},
3         {"cr":800, "m":740, "w":790}]
4 # usage
5 print SATs[0]           # {"cr":780, "m":790, "w":760}
6 print SATs[0]["cr"]    # 780
```

# More Complicated Data Structure

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

```
1 SAT = {"Jane": {"lastname": "Thompson",
2           "test": [{"cr": 700, "m": 690, "w":710}] },
3         "Mary": {"lastname": "Smith",
4           "test": [{"cr": 780, "m": 790, "w":760},
5                   {"cr": 800, "m": 740, "w":790}] }}
6
7 print SAT["Jane"]
8 # {"test": [{"cr": 700, "m": 690, "w": 710}], "lastname": "Thompson"}
9
10 print SAT["Jane"]["lastname"]      # Thompson
11 print SAT["Jane"]["test"]          # [{"cr": 700, "m": 690, "w":710}]
12 print SAT["Jane"]["test"][0]       # {"cr": 700, "m": 690, "w": 710}
13 print SAT["Jane"]["test"][0]["cr"] # 700
14
15 mary1 = SAT["Mary"]["test"][1]
16 print mary1["cr"]                  # 800
```



# Quiz

- ▶ Make a dictionary of 2012 SAT percentile ranks for the scores from 680 to 700 and for all three subsections. The full table is available at <http://tinyurl.com/k38xve8>. Given this dictionary, say  $D$ , a lookup,  $D[680][\text{"cr"}]$  should be evaluated to 93.

# Quiz

- ▶ Make a dictionary of 2012 SAT percentile ranks for the scores from 680 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[680]["cr"]` should be evaluated to 93.
- ▶ An answer.

```
1 D = {700: {"cr": 95, "m": 93, "w": 96},
2       690: {"cr": 94, "m": 92, "w": 95},
3       680: {"cr": 93, "m": 90, "w": 94}}
4
5 print D[680]["cr"]      # 93
```

## Quiz (cont.)

- ▶ (tricky) Write a new dictionary `DD` such that we look up the subsection first and then the score. That is, `DD["cr"][680]` should be evaluated to 93.

(Hint: Start with a dictionary below.):

```
1 DD = {"cr": {700: 95, 690: 94, 680: 93},
2       "m": {700: 93, 690: 92, 680: 90},
3       "w": {700: 96, 690: 95, 680: 94}}
```

# Tuples

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

# Tuples

- ▶ A sequence of values separated by commas. Immutable.

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

- ▶ Often automatically unpacked.

```
1 T = (2, 3)
2 a, b = T             # a is 2, b is 3
3 a, b = b, a         # a and b swapped.
4
5 D = {"x": 23, "y": 46}
6 D.items()           # [("y", 46), ("x", 23)]
7 for k, v in D.items():
8     print "%s ==> %d" % (k, v)   # y ==> 46
9                                     # 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.

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

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

```
1 with open("test.txt", "w") as my_file:
2     my_file.write("first line\n")
3     my_file.write("second line\n")
4     my_file.write("third line")
5
6     print type(my_file)           # <type "file">
7     print dir(my_file)           # properties and methods
8
9 my_file.write("something")        # error. I/O on closed file
```

# Class

- ▶ Let's create a bank account type.

```
1 class BankAccount:
2
3     def __init__(self, initial_balance=0):
4         self.balance = initial_balance
5
6     def deposit(self, amount):
7         self.balance += amount
8
9     def withdraw(self, amount):
10        self.balance -= amount
```

- ▶ Usage examples.

```
1 my_account = BankAccount(100)
2 my_account.withdraw(5)
3 print my_account.balance           # 95
4
5 your_account = BankAccount()
6 your_account.deposit(100)
7 your_account.deposit(10)
8 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.

```
1 class Person:
2     def __init__(self, first, last, year):
3         self.first_name = first_name
4         self.last_name = last_name
5         self.birth_year = birth_year
6
7     def full_name(self):
8         return self.first_name + " " + self.last_name
9
10    def age(self, current_year):
11        return current_year - self.birth_year
12
13    # check
14    mr_park = Person("Jae-sang", "Park", 1977)
15    print mr_park.full_name()           # Jae-sang Park
16    print mr_park.age(2014)            # 37
```

# Inheritance

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

```
1 import webbrowser
2
3 class CoolPerson(Person):
4     def __init__(self, name, birth_year, video):
5         Person.__init__(self, name, None, birth_year)
6         self.video = video
7     def full_name(self):
8         return self.first_name
9     def show_off(self):
10        url = "http://www.youtube.com/watch?v={0}"
11        webbrowser.open(url.format(self.video))
12
13 # check
14 psy = CoolPerson("PSY", 1977, "9bZkp7q19f0")
15 print psy.full_name()           # PSY
16 print psy.age(2012)            # 35
17 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.

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

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

```
1 L = [0, 1, 2, 3]
2 try:
3     print L[5]
4
5 except IndexError:
6     print "no such element"
7
8 print "next"
9 # no such element
10 # next
```

# Throwing Exception

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

```
1 def fetch(a_list, index):
2     if index >= len(a_list):
3         raise IndexError("Uh, oh!")
4     return a_list[index]
5
6 print fetch(L, 5)
7 # IndexError: Uh, oh!
```

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

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

# An Example

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

```
1 import urllib2
2
3 L = ["http://google.com",
4      "http://google.com/somethingfantastic",
5      "http://yahoo.com"]
6
7 # we want to open each page in turn
8 for url in L:
9     try:
10         page = urllib2.urlopen(url)
11         print page.getcode()
12     except urllib2.HTTPError:
13         print "failed to open: {0}".format(url)
14
15 # 200 (a return code of 200 means OK)
16 # failed to open: http://google.com/somethingfantastic
17 # 200
```

# 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.