Lecture 6. Python Primer – Part 6

Chao Chen

Stony Brook University

Oct. 24, 2022

Tuple

- Immutable, sequence
- Initialization

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

- Creating one-element tuples

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

```
>>> t2 = ('a')
>>> type(t2)
<type 'str'>
```

[1]Fun fact: The word “tuple” comes from the names given to sequences of numbers of varying lengths: single, double, triple, quadruple, quituple, sextuple, septuple, etc.

Tuple

- Creating tuples (continued)

```
>>> t = tuple()
>>> print(t)
()
```

```
>>> t = tuple('lupins')
>>> print(t)
('l', 'u', 'p', 'i', 'n', 's')
```

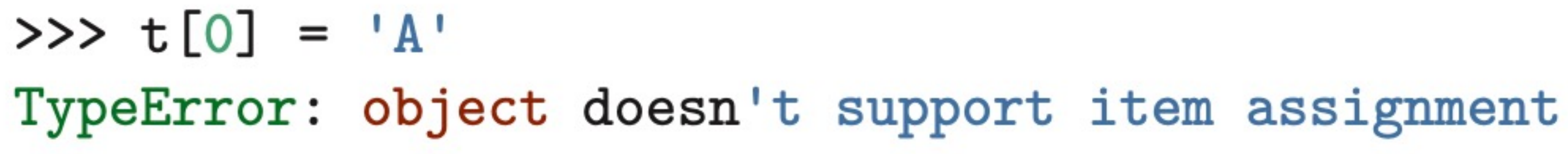
- Accessing (mostly like list)

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print(t[0])
'a'
```

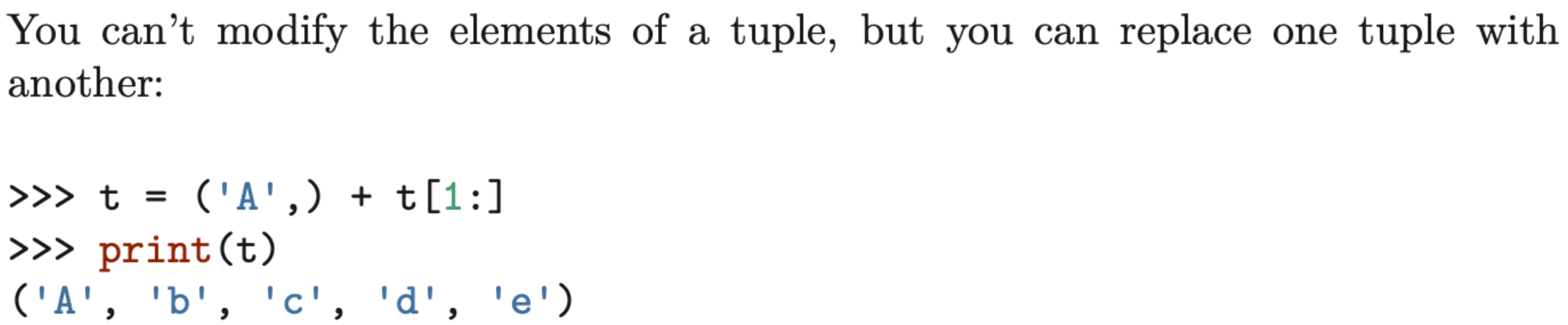
```
>>> print(t[1:3])
('b', 'c')
```

Tuple

- Immutable – cannot modify its content



```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```



You can't modify the elements of a tuple, but you can replace one tuple with another:

```
>>> t = ('A',) + t[1:]
>>> print(t)
('A', 'b', 'c', 'd', 'e')
```

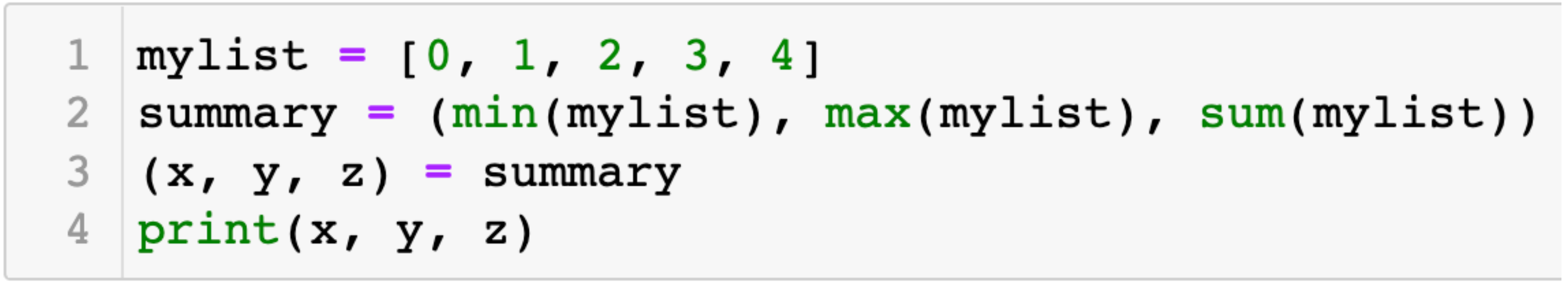
Tuple

- Comparison
  - Comparing the first element, then the second, etc.

```
>>> (0, 1, 2) < (0, 3, 4)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
```

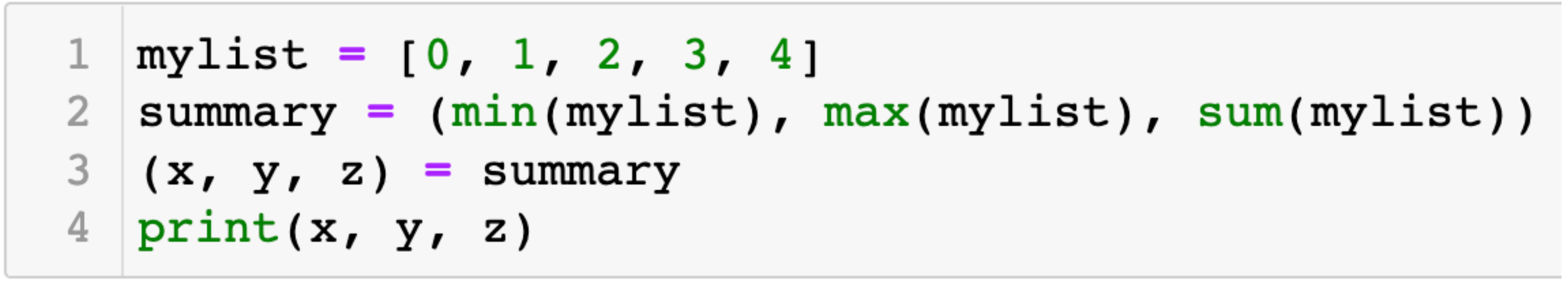
Tuple

- Assignment from tuple to tuple



```python
mylist = [0, 1, 2, 3, 4]
summary = (min(mylist), max(mylist), sum(mylist))
(x, y, z) = summary
print(x, y, z)
```

```
0 4 10
```

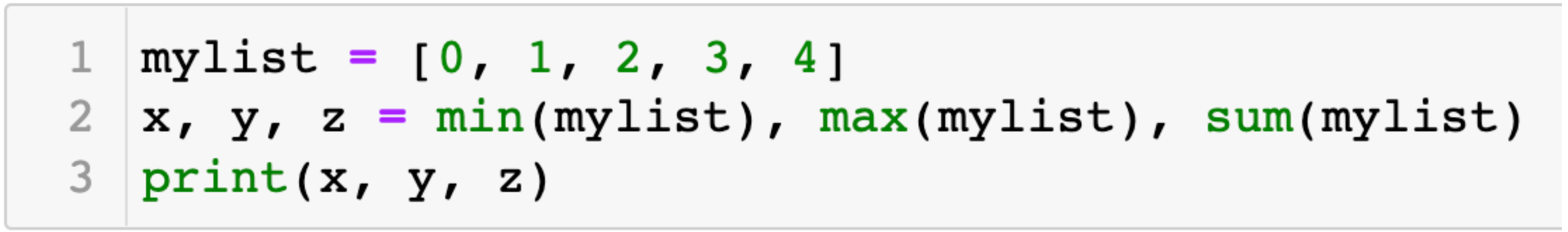


```python
mylist = [0, 1, 2, 3, 4]
summary = (min(mylist), max(mylist), sum(mylist))
(x, y, z) = summary
print(x, y, z)
```

```
0 4 10
```

Tuple

- Parenthesis is often omitted



```
mylist = [0, 1, 2, 3, 4]
x, y, z = min(mylist), max(mylist), sum(mylist)
print(x, y, z)
```

```
0 4 10
```

- # of elements on left and right have to match

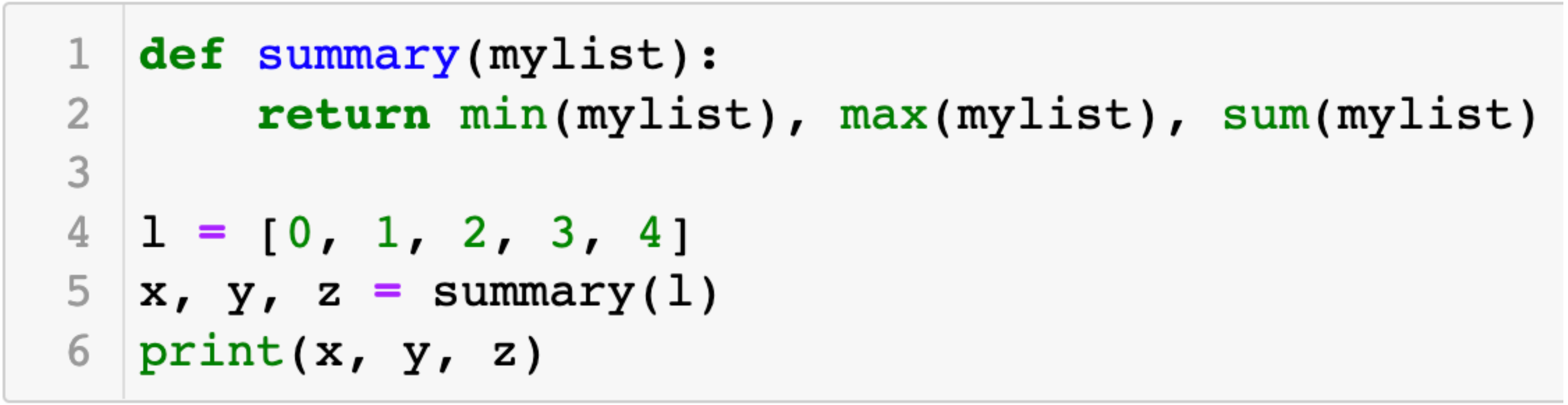
```
>>> a, b = 1, 2, 3
ValueError: too many values to unpack
```

Tuple

- An easy way to swap values of two variables (not possible in C++/java)

```
>>> a, b = b, a
```

- Useful when a function returns multiple values



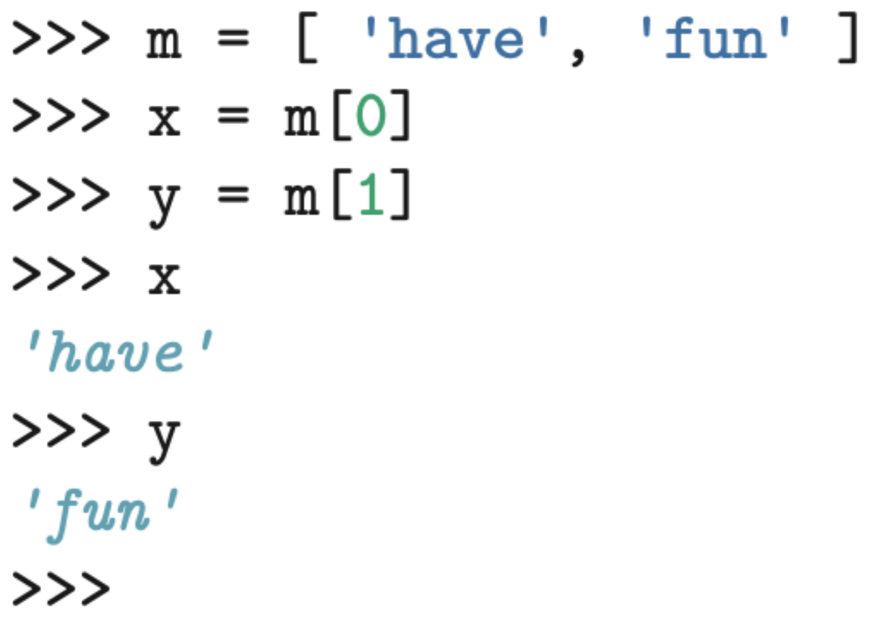
```
def summary(mylist):
    return min(mylist), max(mylist), sum(mylist)

l = [0, 1, 2, 3, 4]
x, y, z = summary(l)
print(x, y, z)
```

```
0 4 10
```

Tuple

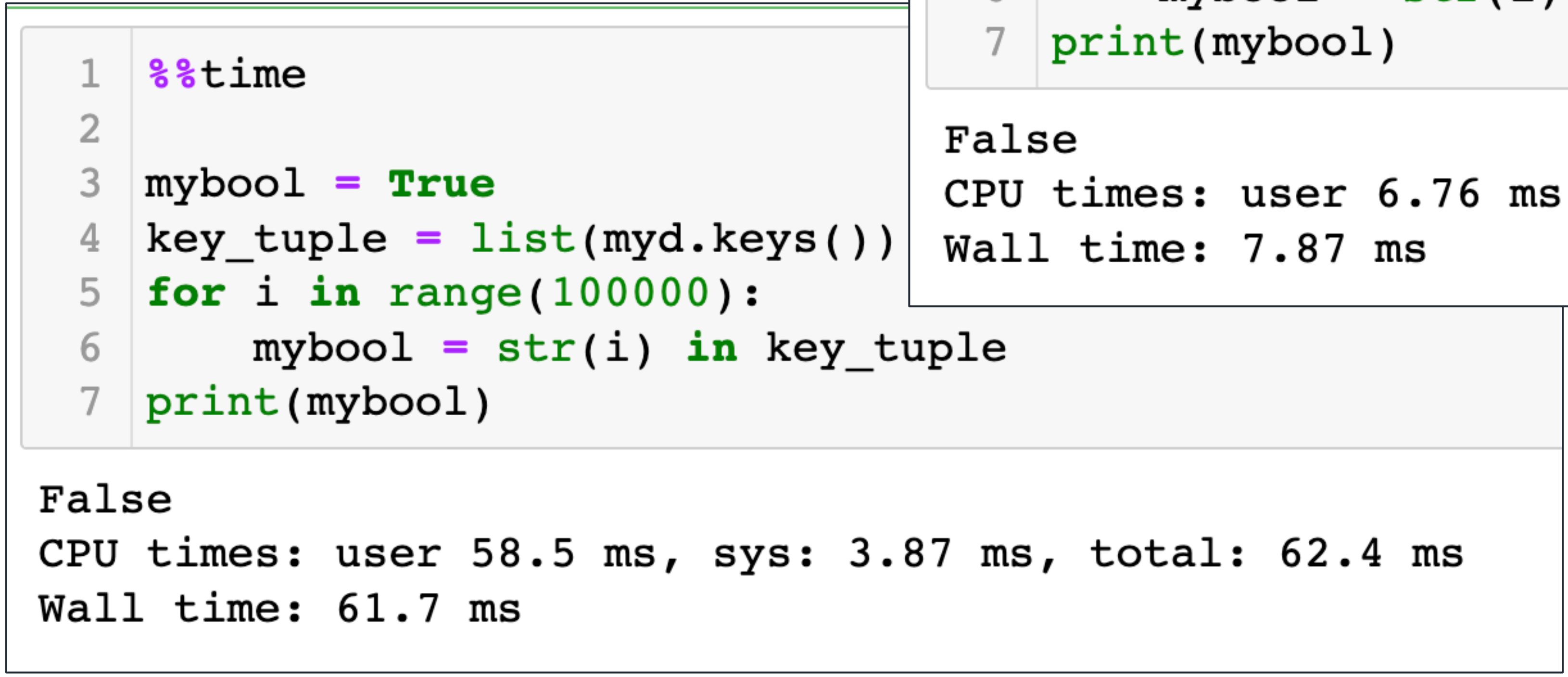
- Right hand side can also be a list
- But this is not recommended (conceptually confusing to me)



```
>>> m = [ 'have', 'fun' ]
>>> x = m[0]
>>> y = m[1]
>>> x
'have'
>>> y
'fun'
>>>
```

Tuple

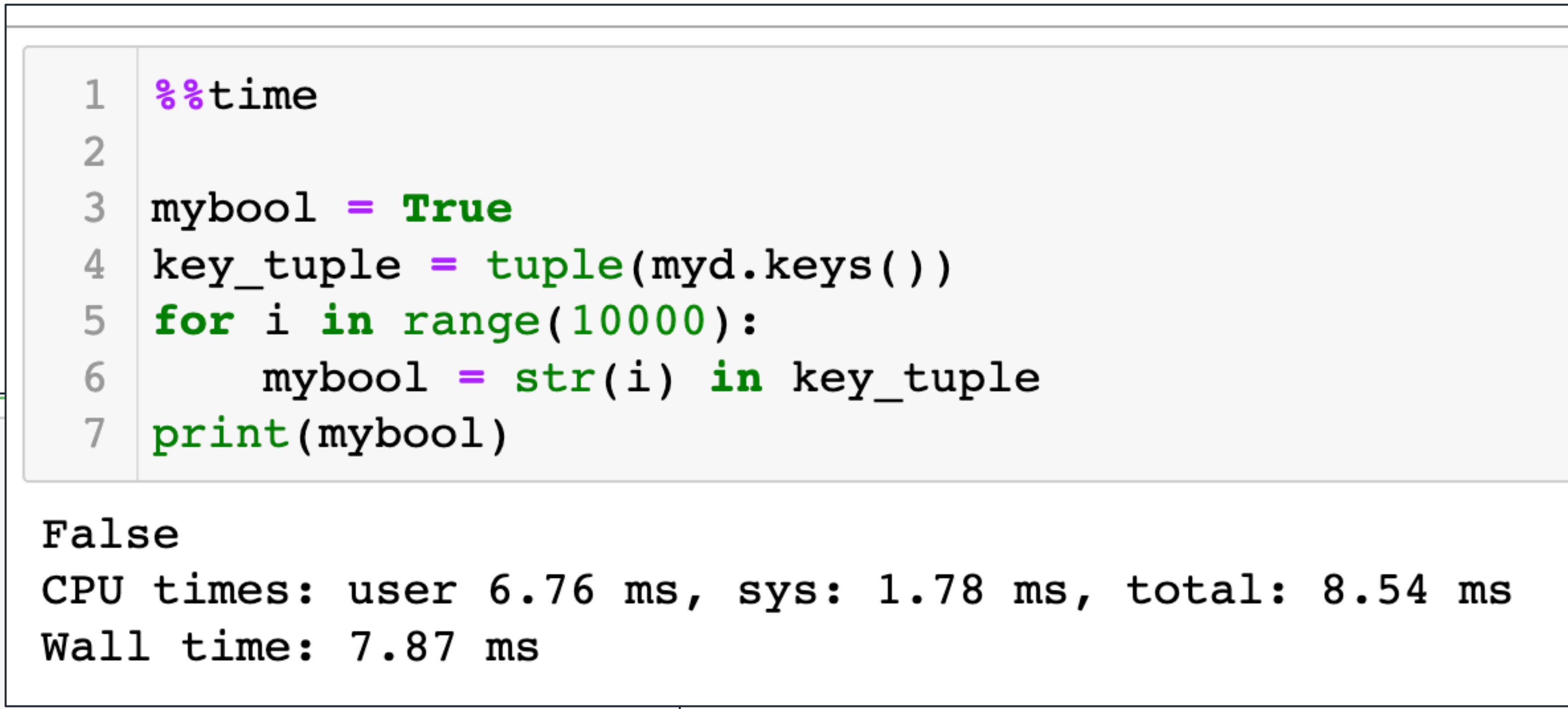
- Benefit of tuple compared to list -- efficiency



```
%%time

mybool = True
key_tuple = list(myd.keys())
for i in range(100000):
    mybool = str(i) in key_tuple
print(mybool)
```

```
False
CPU times: user 58.5 ms, sys: 3.87 ms, total: 62.4 ms
Wall time: 61.7 ms
```



```
%%time

mybool = True
key_tuple = tuple(myd.keys())
for i in range(10000):
    mybool = str(i) in key_tuple
print(mybool)
```

```
False
CPU times: user 6.76 ms, sys: 1.78 ms, total: 8.54 ms
Wall time: 7.87 ms
```

“in” – scalability issue revisited

- range(10000)
  in keys – 6.3 ms in keys_tuple – 7.87 ms
  in keys_list – 780 ms
- range(100000)
  in keys – 50.6 ms in keys_tuple – 61.7 ms
  in keys_list – 1min 19s = 79,000 ms
- keys and keys_tuple – implemented using hash table
  constant operation for each ”
- keys_list – linear operation
  can be as expensive as the lis

Tuple is more efficient than list.
What’s the catch – flexibility!

Tuple and dictionary

- Using items() function of a dictionary, getting a collection of tuples, each tuple – (key, val)

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = list(d.items())
>>> print(t)
[('b', 1), ('a', 10), ('c', 22)]
```

```
for key, val in list(d.items()):
    print(val, key)
```

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = list(d.items())
>>> t
[('b', 1), ('a', 10), ('c', 22)]
>>> t.sort()
>>> t
[('a', 10), ('b', 1), ('c', 22)]
```

```python
import string
fhand = open('romeo-full.txt')
counts = dict()
for line in fhand:
    line = line.translate(string.punctuation)
    line = line.lower()
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

# Sort the dictionary by value
lst = list()
for key, val in list(counts.items()):
    lst.append((val, key))

lst.sort(reverse=True)

for key, val in lst[:10]:
    print(key, val)
```

```
61 i
42 and
40 romeo
34 to
34 the
32 thou
32 juliet
30 that
29 my
24 thee
```

Tuple

1. In some contexts, like a `return` statement, it is syntactically simpler to create a tuple than a list. In other contexts, you might prefer a list.

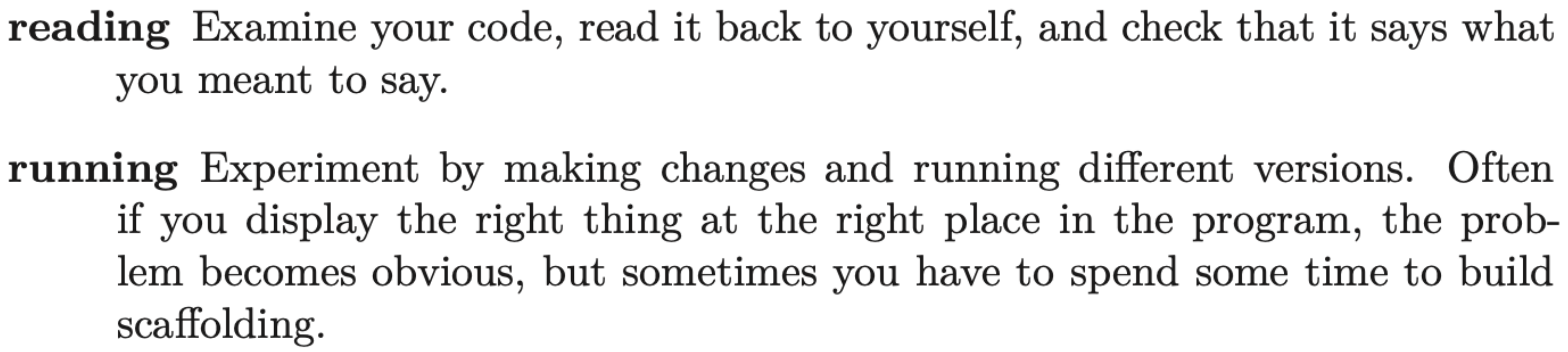
2. If you want to use a sequence as a dictionary key, you have to use an immutable type like a tuple or string.

3. If you are passing a sequence as an argument to a function, using tuples reduces the potential for unexpected behavior due to aliasing.

Because tuples are immutable, they don't provide methods like `sort` and `reverse`, which modify existing lists. However Python provides the built-in functions `sorted` and `reversed`, which take any sequence as a parameter and return a new sequence with the same elements in a different order.

Tuple

- List, dictionaries, tuples --- data structures
- Shape error – wrong type, size, composition, wrong shape
- Debugging tips

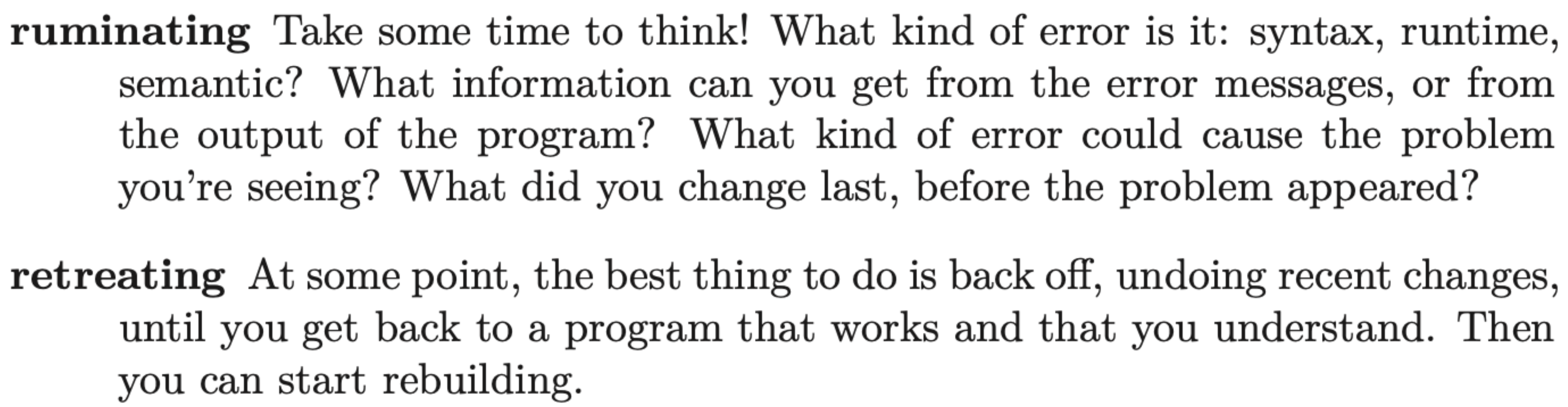


reading Examine your code, read it back to yourself, and check that it says what you meant to say.

running Experiment by making changes and running different versions. Often if you display the right thing at the right place in the program, the problem becomes obvious, but sometimes you have to spend some time to build scaffolding.

Tuple

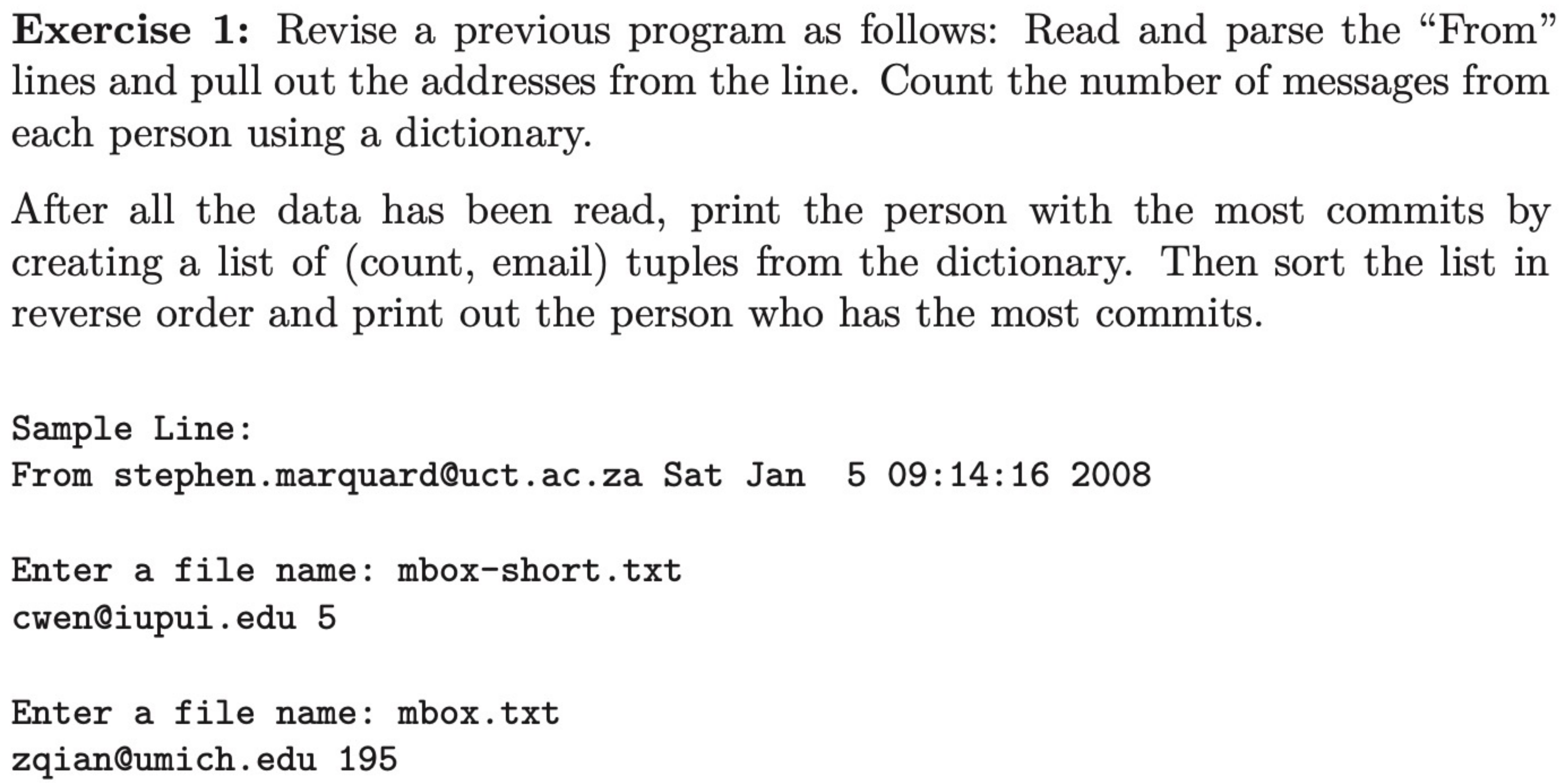
- Debugging tips (cont’d)



ruminating Take some time to think! What kind of error is it: syntax, runtime, semantic? What information can you get from the error messages, or from the output of the program? What kind of error could cause the problem you’re seeing? What did you change last, before the problem appeared?

retreating At some point, the best thing to do is back off, undoing recent changes, until you get back to a program that works and that you understand. Then you can start rebuilding.

Tuple



Exercise 1: Revise a previous program as follows: Read and parse the "From" lines and pull out the addresses from the line. Count the number of messages from each person using a dictionary.

After all the data has been read, print the person with the most commits by creating a list of (count, email) tuples from the dictionary. Then sort the list in reverse order and print out the person who has the most commits.

```
Sample Line:
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

Enter a file name: mbox-short.txt
cwen@iupui.edu 5

Enter a file name: mbox.txt
zqian@umich.edu 195
```

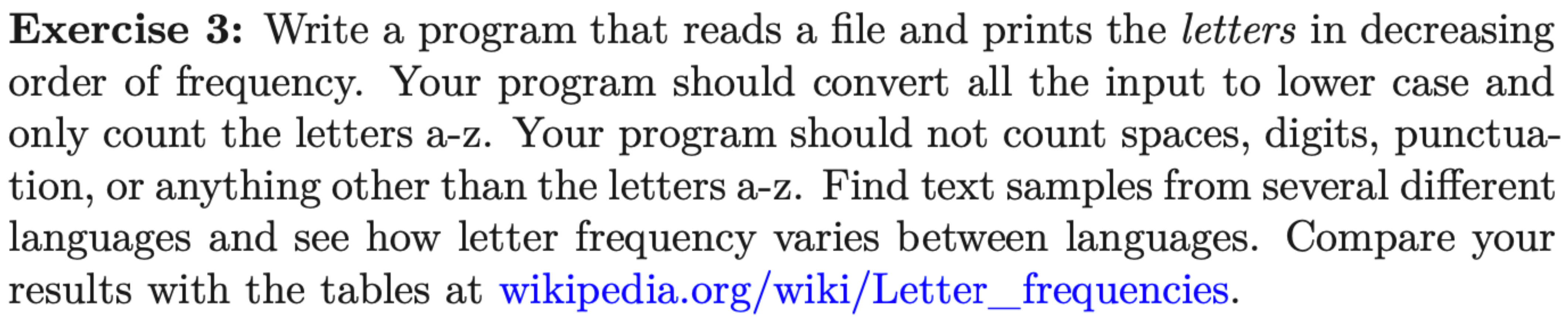
Exercise 2: This program counts the distribution of the hour of the day for each of the messages. You can pull the hour from the "From" line by finding the time string and then splitting that string into parts using the colon character. Once you have accumulated the counts for each hour, print out the counts, one per line, sorted by hour as shown below.

Sample Execution:

```
python timeofday.py
Enter a file name: mbox-short.txt
04 3
06 1
07 1
09 2

10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1
```

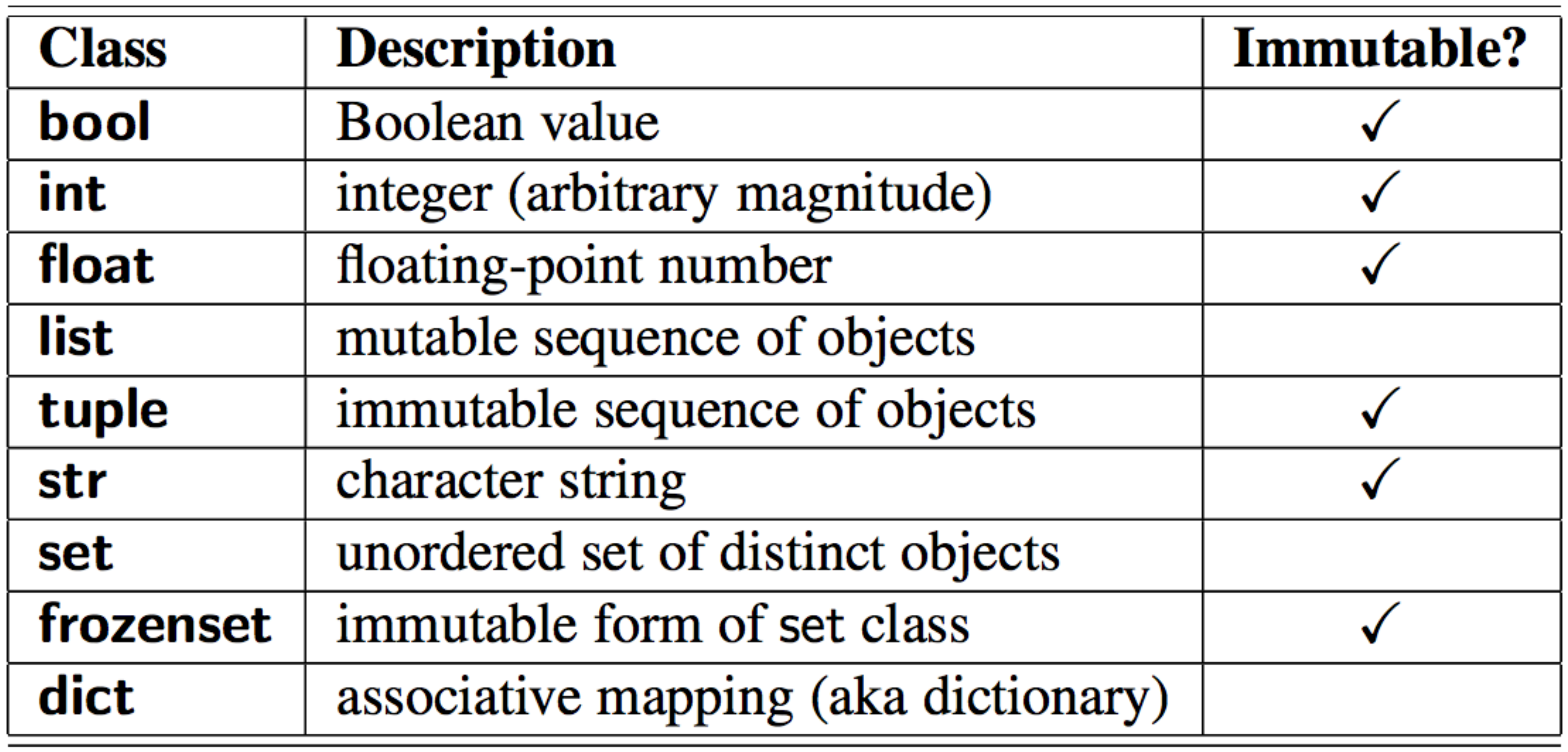
Tuple



Exercise 3: Write a program that reads a file and prints the *letters* in decreasing order of frequency. Your program should convert all the input to lower case and only count the letters a-z. Your program should not count spaces, digits, punctuation, or anything other than the letters a-z. Find text samples from several different languages and see how letter frequency varies between languages. Compare your results with the tables at wikipedia.org/wiki/Letter_frequencies.

Built-in Classes

- Immutable: object value cannot be changed
- Identifier can be reassigned



Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

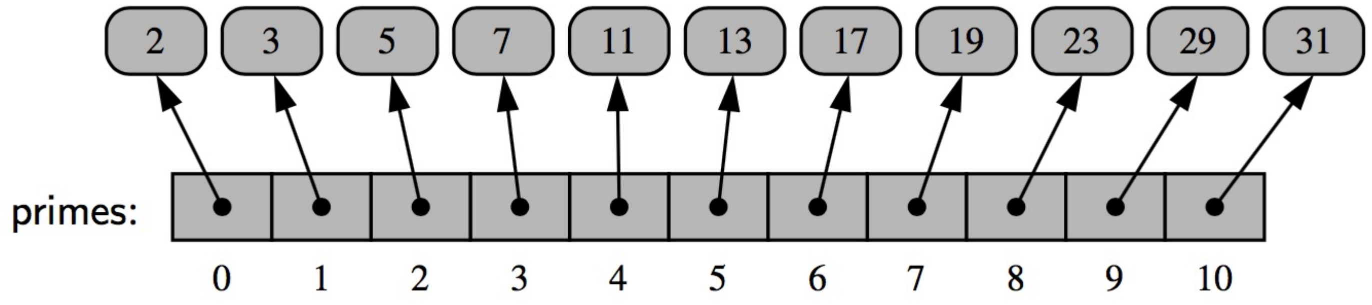
Table 1.2: Commonly used built-in classes for Python

Built-in Classes (Cont'd)

- bool:
  - Values: True or False; bool() returns False; bool(val), with val from other types
- int (automatically choose internal representation based on magnitude):
  - initiate by int values: myInt = 10
  - Binary, octal and hexadecimal: 0b1011, 0o52, 0x7f (base of 2, 8 and 16)
  - int() returns 0
  - int(val) return truncated value for val being float: int(3.4), int(3.99), int(-3.9)
  - int('137') converts a string to int if possible
  - int('7f', 16) converts from a different base to decimal
  - Decimal to other bases: bin(…), oct(…), hex(…)
- float (close to double in Java and C/C++):
  - myFloat = 8.9 myFloat = 8. myFloat = .8 myFloat = 6.022e23 myFloat = float()
  - myFloat = float(2) myFloat = float('3.14')
  - sys.float_info
- type(…) – check the type of a variable

Built-in Classes (Cont'd)

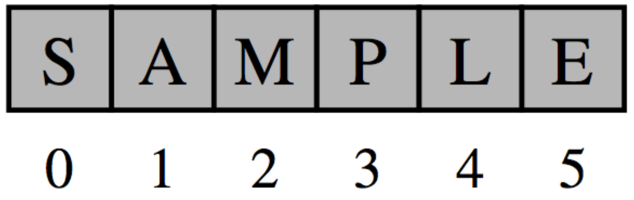
- Sequence classes (list, tuple, str): A collection of values with (important) ordering
- list (mutable)
  - referential: stores an array of references to objects



  - Zero-Indexed: primes[0], ..., primes[len(primes)-1]
  - Can be a mixture of (arbitrary) types, init using values/references
    v_str = 'tmp str'; v_float = 3.14
    myList = [3, v_str, v_float, 'tmp str again']
  - Init using an empty list: myList = []
  - Issue: list of lists, aliases for entries, be careful! Not an issue for basic types.
- Disclaimer: when copying code from this slide, the " ' " symbol can be problematic.

Built-in Classes (Cont’d)

- tuple:
  - Immutable version of list, use ‘()’ instead of ‘[]’
  - Once initialized, cannot change values
  - For single element tuple, use myTuple = (17,) instead of myTuple = (17), why?
- str (also immutable):
  - myStr = ’sample’ or “sample”
  - myStr = ”Don’t worry” or ‘Don\’t worry’
  - myStr = ‘C:\\Python\\’   other special chars: ’\n’ – line break; ‘\t’ – tab
  - Use ’’’ or ””” to begin/end a a string literally.



```
print("""Welcome to the GPA calculator.
Please enter all your letter grades, one per line.
Enter a blank line to designate the end.""")
```

Built-in Classes (Cont’d)

- set:
  - A set of elements without ordering (no repeating elements)
  - Implemented using hash table (will talk in the future)
  - Only contains immutables as values (no sets or lists as values)
  - Immutable version – frozenset
  - Use curly braces ’ { } ’
  - Empty set: use set(), not {} – reserved for empty dictionary
  - { ‘red’ , ‘green’ , ‘blue’ }
  - Constructor: convert an iterable input into a set of its element
    mySet = set(‘hello’) is equivalent to mySet = { ‘h’ , ‘e’ , ‘l’, ‘o’ }.
    Why one less ‘l’?

Expressions and Operators (Cont'd)

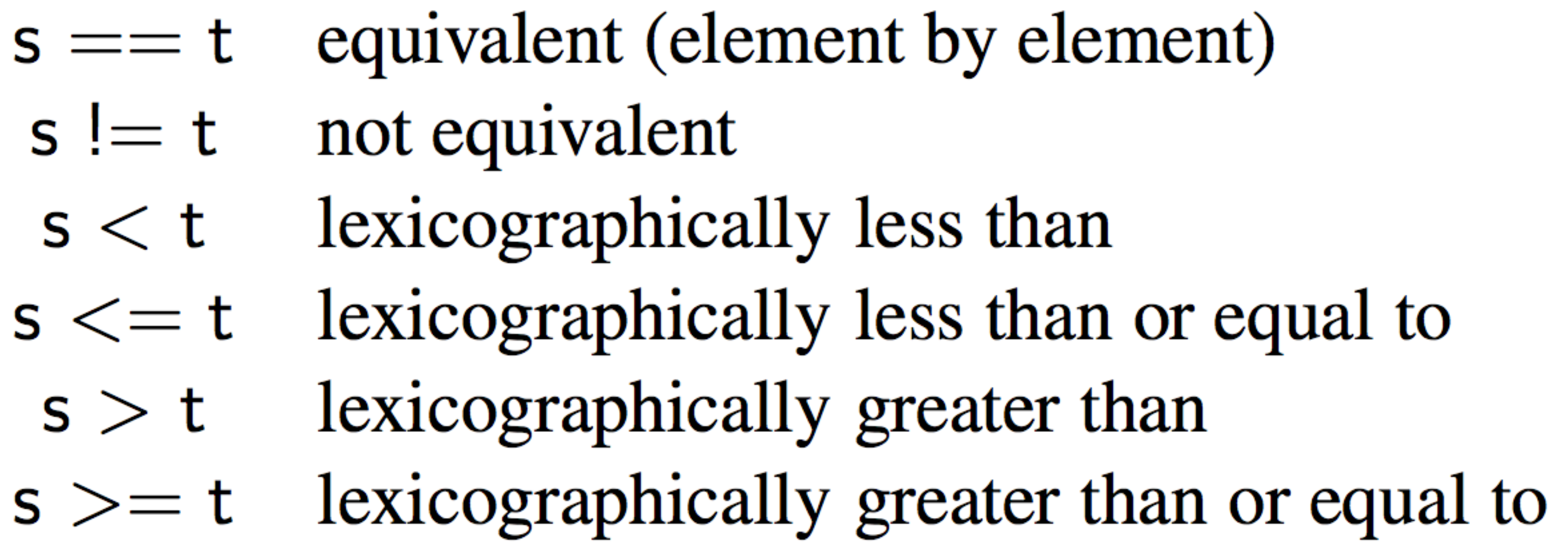
- **Sequence operators (tuple, str, list):**

s[j]	element at index *j*
s[start:stop]	slice including indices [start,stop)
s[start:stop:step]	slice including indices start, start + step, start + 2∗step, ..., up to but not equalling or stop
s + t	concatenation of sequences
k ∗ s	shorthand for s + s + s + ... (k times)
val **in** s	containment check
val **not in** s	non-containment check

- Indexing: 0 to n-1, -1 = n-1, -2 = n-2; Slice: half-open interval of idx
- For lists: s[i] = new_val; **del** s[i] → remove i-th entry what if s[i] has an alias?
- Same thing (replace or delete) can be done on a sublist via slicing.
- For string, 'amp' in 'example' Q: what will happen with [2,3,4] in [1,2,3,4,5]

Expressions and Operators (Cont'd)

- **Sequence comparison (tuple, str, list):**

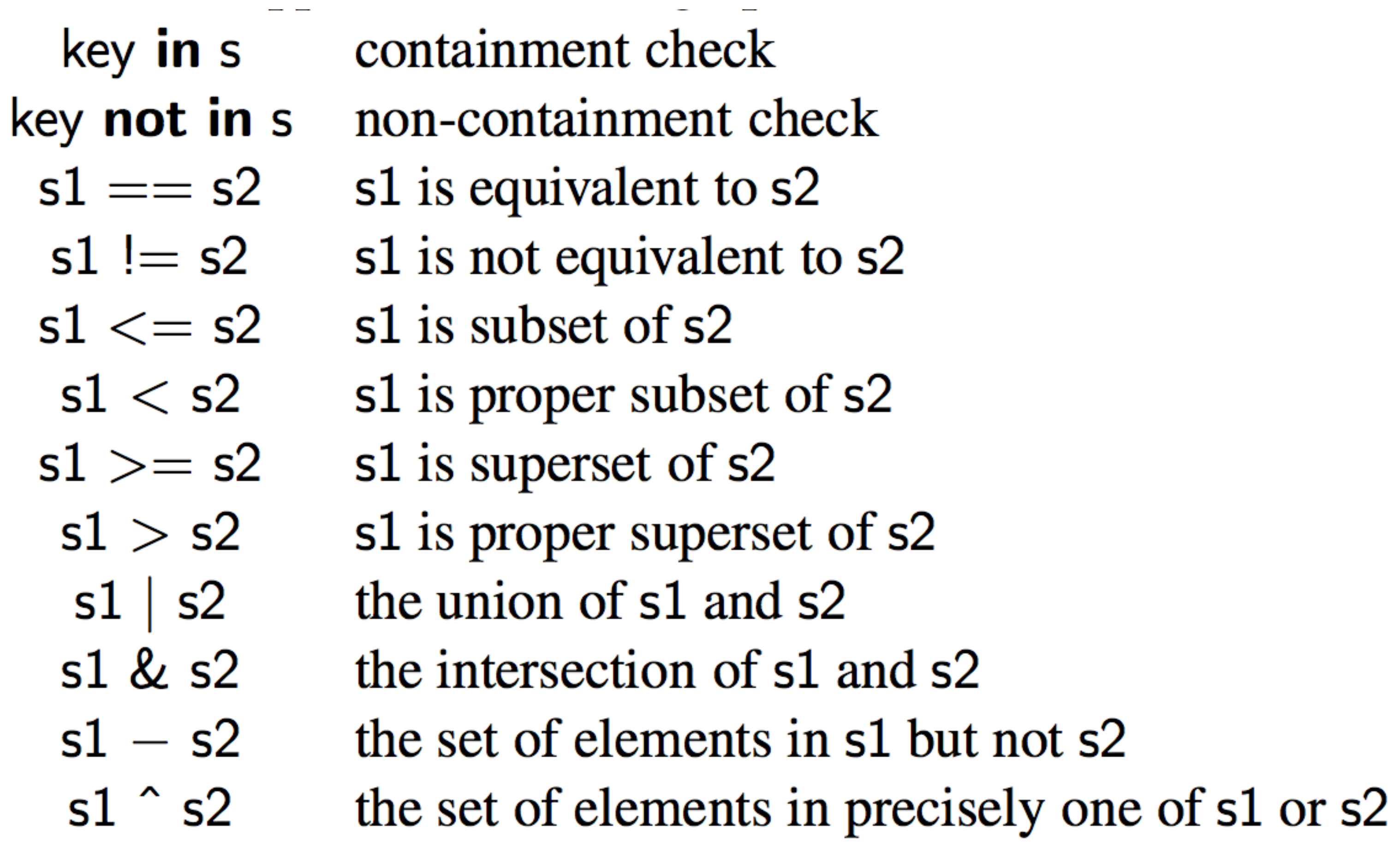


s == t	equivalent (element by element)
s != t	not equivalent
s < t	lexicographically less than
s <= t	lexicographically less than or equal to
s > t	lexicographically greater than
s >= t	lexicographically greater than or equal to

- Lexicographical, length does not count
- [5,6,9] < [5,7,1] – True or False?   [5,7,-9] < [5,7] -- True or False?

Expressions and Operators (Cont’d)

- **Set/frozenset operators:**



key **in** s	containment check
key **not in** s	non-containment check
s1 == s2	s1 is equivalent to s2
s1 != s2	s1 is not equivalent to s2
s1 <= s2	s1 is subset of s2
s1 < s2	s1 is proper subset of s2
s1 >= s2	s1 is superset of s2
s1 > s2	s1 is proper superset of s2
s1 | s2	the union of s1 and s2
s1 & s2	the intersection of s1 and s2
s1 − s2	the set of elements in s1 but not s2
s1 ˆ s2	the set of elements in precisely one of s1 or s2

  - S1 ^ S2 == (S1 – S2) | (S2-S1) == (S1 | S2) – (S1 & S2)
  - Set also supports basic operations like add, remove, etc (will cover in the future)

Expressions and Operators (Cont'd)

- **dict operators:**

d[key]	value associated with given key
d[key] = value	set (or reset) the value associated with given key
del d[key]	remove key and its associated value from dictionary
key **in** d	containment check
key **not in** d	non-containment check
d1 == d2	d1 is equivalent to d2
d1 != d2	d1 is not equivalent to d2

- Comparison, e.g., <, is invalid.
- d1 == d2 if and only if d1 and d2 contain the same set of keys and also same values

Expressions and Operators (Cont'd)

- **Extended assignment:**
  - For immutables: count += 5 --> allocate a new object
  - For lists:

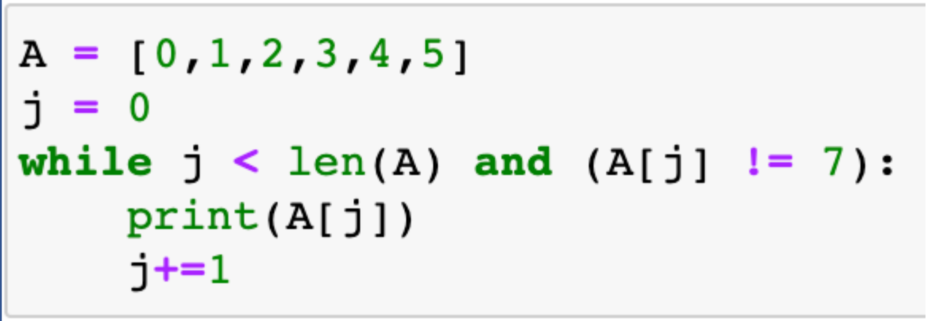
```
alpha = [1, 2, 3]
beta = alpha                # an alias for alpha
beta += [4, 5]              # extends the original list with two more elements
beta = beta + [6, 7]        # reassigns beta to a new list [1, 2, 3, 4, 5, 6, 7]
print(alpha)                # will be [1, 2, 3, 4, 5]
```

Loops

while *condition*:
    body

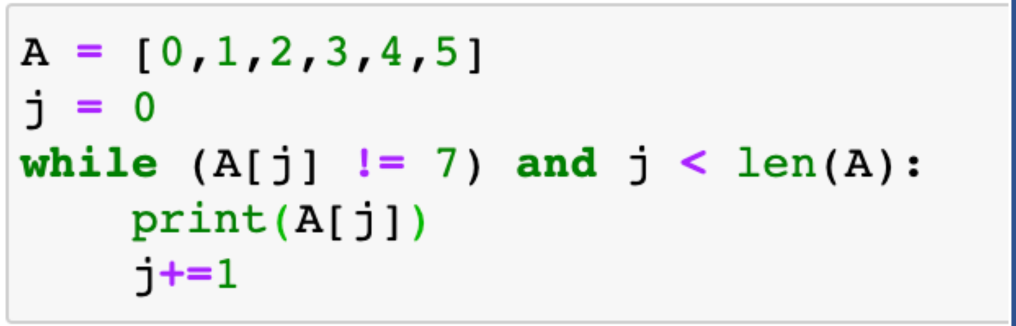
- What is the outcome when A = [0,1,2,7,4,5]

What does this code do?



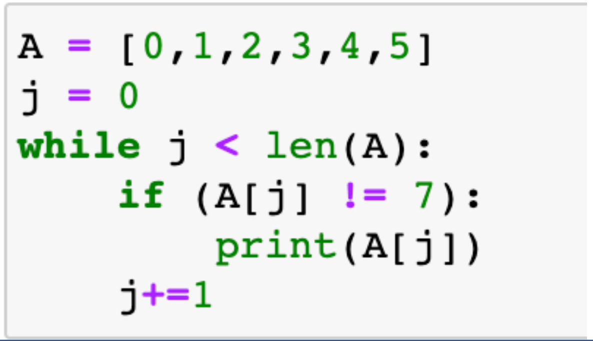
```python
A = [0,1,2,3,4,5]
j = 0
while j < len(A) and (A[j] != 7):
    print(A[j])
    j+=1
```

Why does this code crash?



```python
A = [0,1,2,3,4,5]
j = 0
while (A[j] != 7) and j < len(A):
    print(A[j])
    j+=1
```

What does this code do?

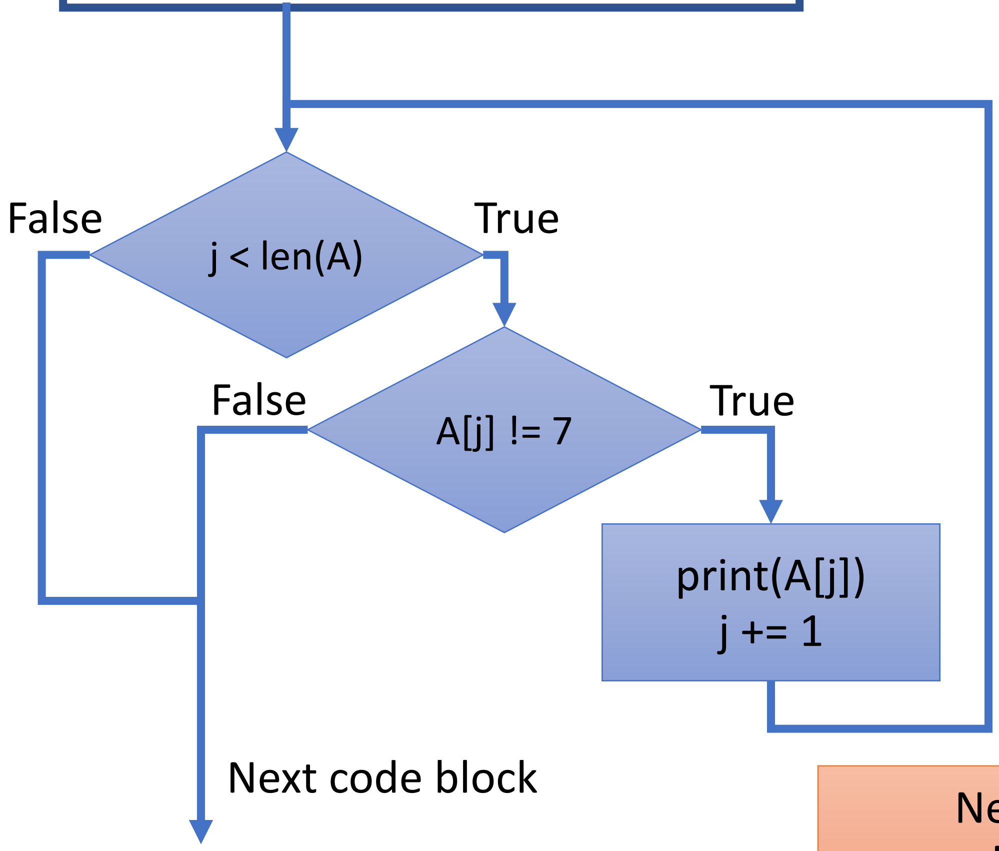


```python
A = [0,1,2,3,4,5]
j = 0
while j < len(A):
    if (A[j] != 7):
        print(A[j])
    j+=1
```

Loops

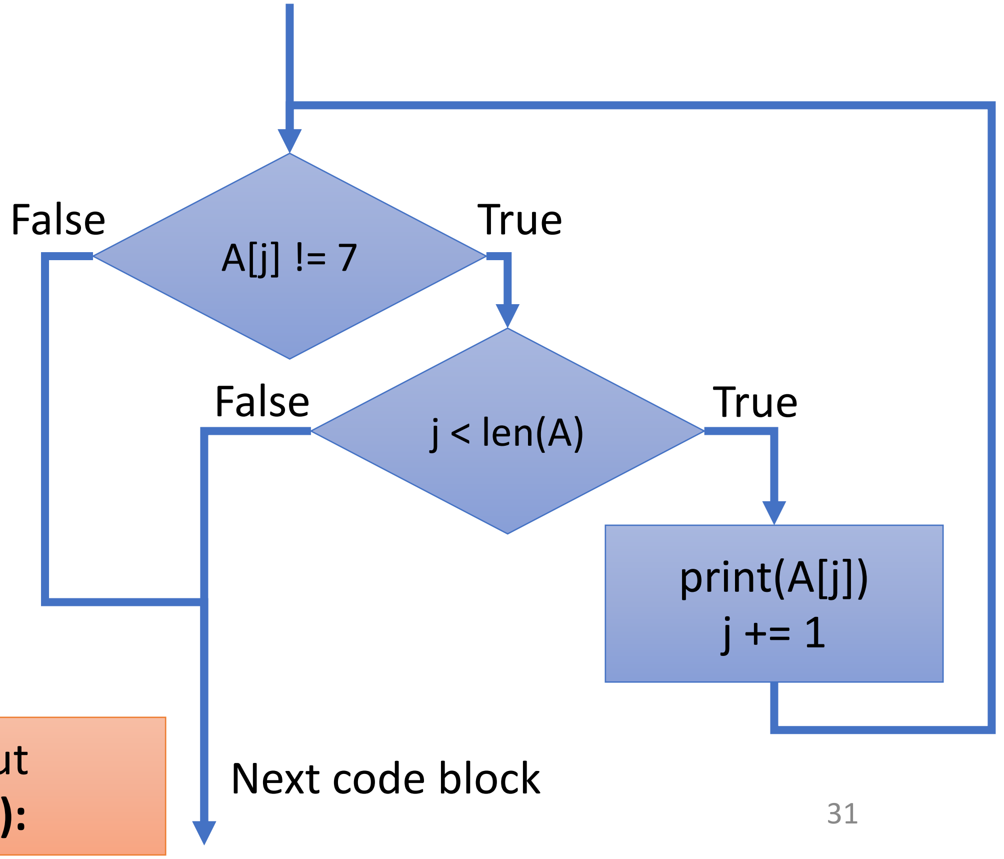
What does this code do?

```
A = [0,1,2,3,4,5]
j = 0
while j < len(A) and (A[j] != 7):
    print(A[j])
    j+=1
```



Why does this code crash?

```
A = [0,1,2,3,4,5]
j = 0
while (A[j] != 7) and j < len(A):
    print(A[j])
    j+=1
```



New Question: what about
while (cond 1) or (cond 2):

Loops (cont'd)

```
for element in iterable:
    body                    # body may refer to 'element' as an identifier
```

Calculating sum of elements in a list

```
total = 0
for val in data:
    total += val
```

Calculating max of elements in a list

```
biggest = data[0]
for val in data:
    if val > biggest:
        biggest = val
```

Q1: Calculating average of elements in a list?

Calculating average of even-valued elements in a list?

Q2: how to get the index of the largest element?

Loops (cont'd)

- **for** loop with explicit indexing

  Example:
  get the largest element's index.

```
big_index = 0
for j in range(len(data)):
  if data[j] > data[big_index]:
    big_index = j
```

- What are range(10), range(1,10), range(1,10,2)?

- Controlling loops
  - **Break:** stop loop and exit
  - **Continue:** skip the rest of the current loop

- Example:
  - checking if a target number exists in the data list
  - Count # of times the target number appears

```
found = False
for item in data:
  if item == target:
    found = True
    break
```

Loops (cont'd)

- Controlling loops
  - **Break:** stop loop and exit
  - **Continue:** skip the rest of the current loop
- What do the following codes do?

```
tot = 0
for item in data:
    if item % 2 == 0:
        break
    tot += item
print(tot)
```

```
tot = 0
for item in data:
    if item % 2 == 0:
        continue
    tot += item
print(tot)
```

- **for** is simpler than **while** in referring to the iterable content
  Q: how to rewrite a for loop using while? When the iterable is a set?

Functions

- Methods (specifically bounded with classes, will talk later)
- First line: signature
  - Has: identifier, parameters (with identifiers)
  - Has not: returning, types of params (as in C, Java)
  - Misuse will only be detected in run time.
- Body (indented block)
  - Inside: namespace – local scope (param names)
- Return: one or multiple.
  if no return line, return None

```
def count(data, target):
  n = 0
  for item in data:
    if item == target:
      n += 1
  return n
```

- To call: `prizes = count(grades, 'A')`

Functions

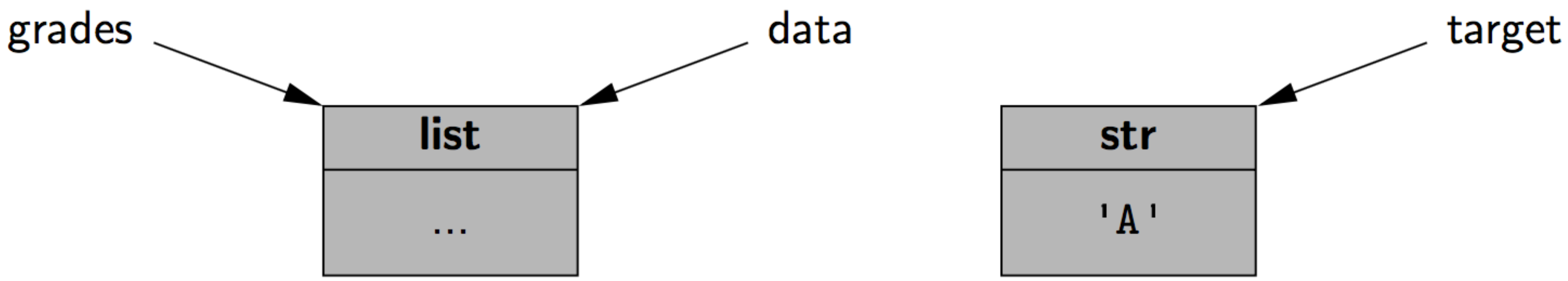
- Information passing:
  - By references: params and returns (In other languages: also by values etc.)
  - Think of it as assignment statements

```
prizes = count(grades, 'A')
```

```
def count(data, target):
  n = 0
  for item in data:
    if item == target:
      n += 1
  return n
```

  - Equivalent to in a local scope

```
data = grades
target = 'A'
```



  - Same for returning: prizes get assigned the object created inside prizes (global) = n (local)

Functions

```
def scale(data, factor):
  for j in range(len(data)):
    data[j] *= factor
```

- Mutable Parameters
  - Can also do data.append('F')
- what happens if we have the line: "data = []" ?
- Default param values
  - ***polymorphic***
  - def foo(a, b=20, c=30):
        return a + b + c
  - Can pass in 1, 2 or 3 params
    print(foo(10)); print(foo(10, 10));
    print(foo(10, 10, 10)); print(foo( ));
    print(foo(a=10)); print(foo(10, c=10));
    print(foo(b=10, c=10))
  - def bar(a, b=15, c) -- illegal

Functions

- Another example

```
def compute_gpa(grades, points={'A+':4.0, 'A':4.0, 'A-':3.67, 'B+':3.33,
                                'B':3.0, 'B-':2.67,'C+':2.33, 'C':2.0,
                                'C':1.67, 'D+':1.33, 'D':1.0, 'F':0.0}):
  num_courses = 0
  total_points = 0
  for g in grades:
    if g in points:                                   # a recognizable grade
      num_courses += 1
      total_points += points[g]
  return total_points / num_courses
```

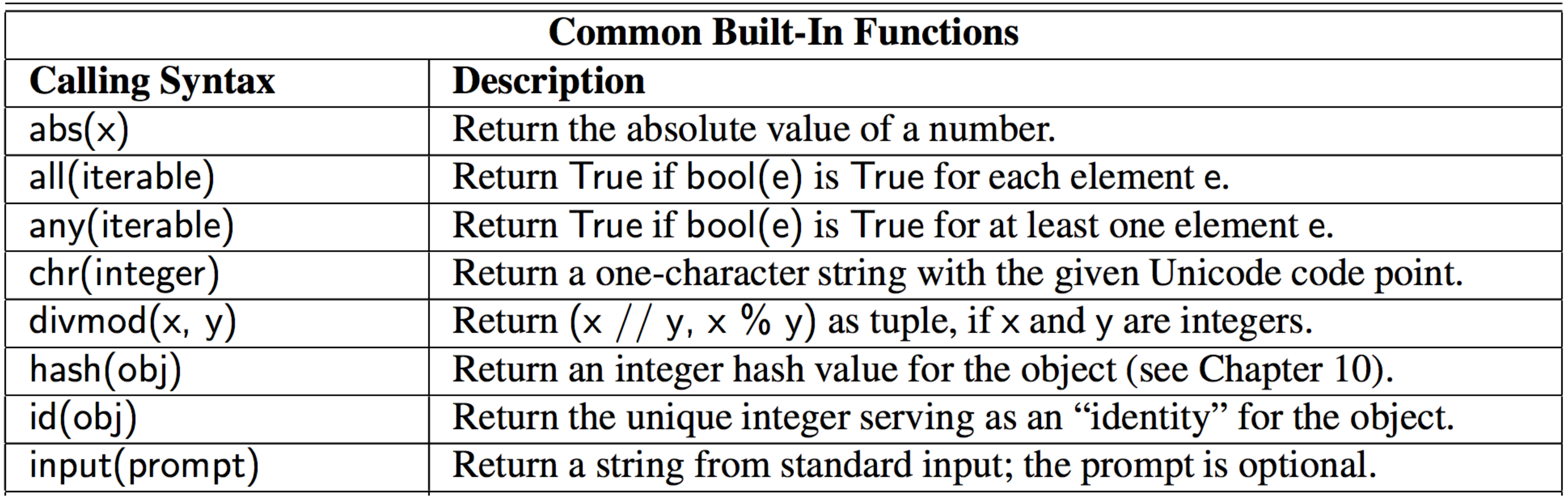
Functions

- Another example: range (three forms)
  range(n); range(start, stop); range(start, stop, step);
- How to implement this?

```
def range(start, stop=None, step=1):
  if stop is None:
     stop = start
     start = 0
  ...
```

- max(a, b, key=abs); max(a, b, c, d);

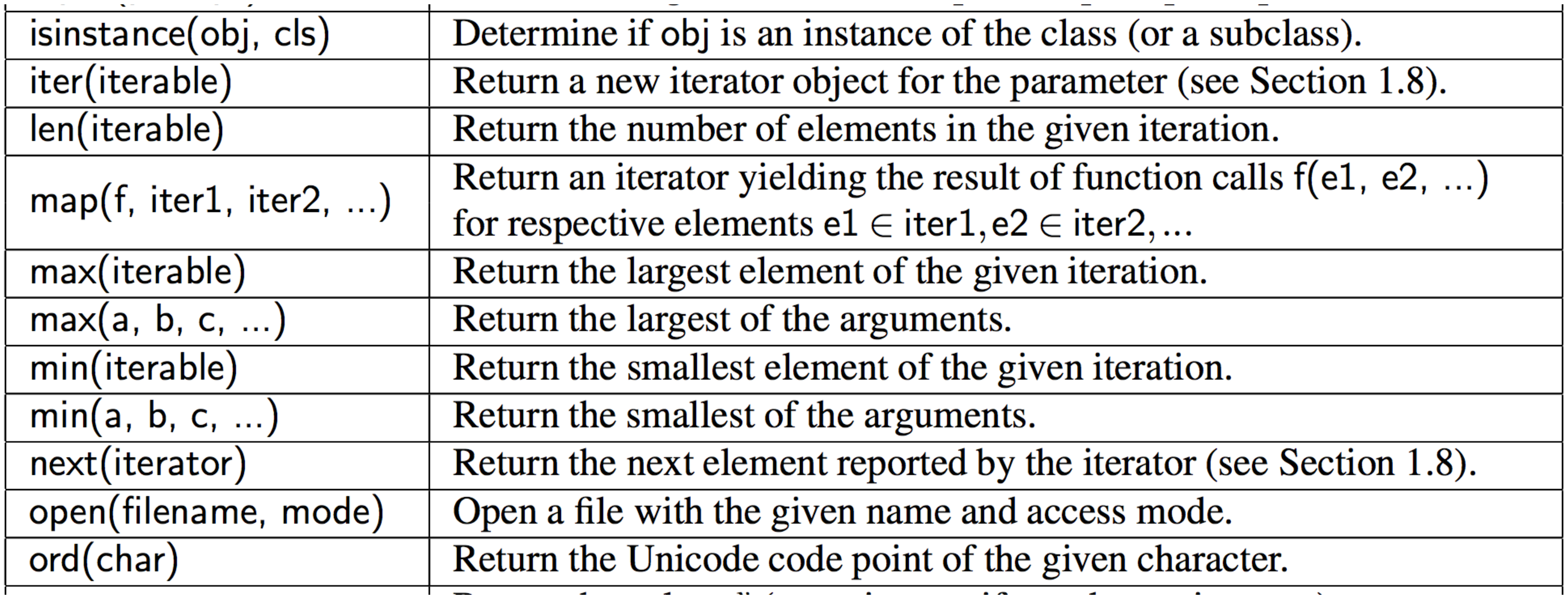
Built-in Functions



Common Built-In Functions

Calling Syntax	**Description**
abs(x)	Return the absolute value of a number.
all(iterable)	Return True if bool(e) is True for each element e.
any(iterable)	Return True if bool(e) is True for at least one element e.
chr(integer)	Return a one-character string with the given Unicode code point.
divmod(x, y)	Return (x // y, x % y) as tuple, if x and y are integers.
hash(obj)	Return an integer hash value for the object (see Chapter 10).
id(obj)	Return the unique integer serving as an “identity” for the object.
input(prompt)	Return a string from standard input; the prompt is optional.

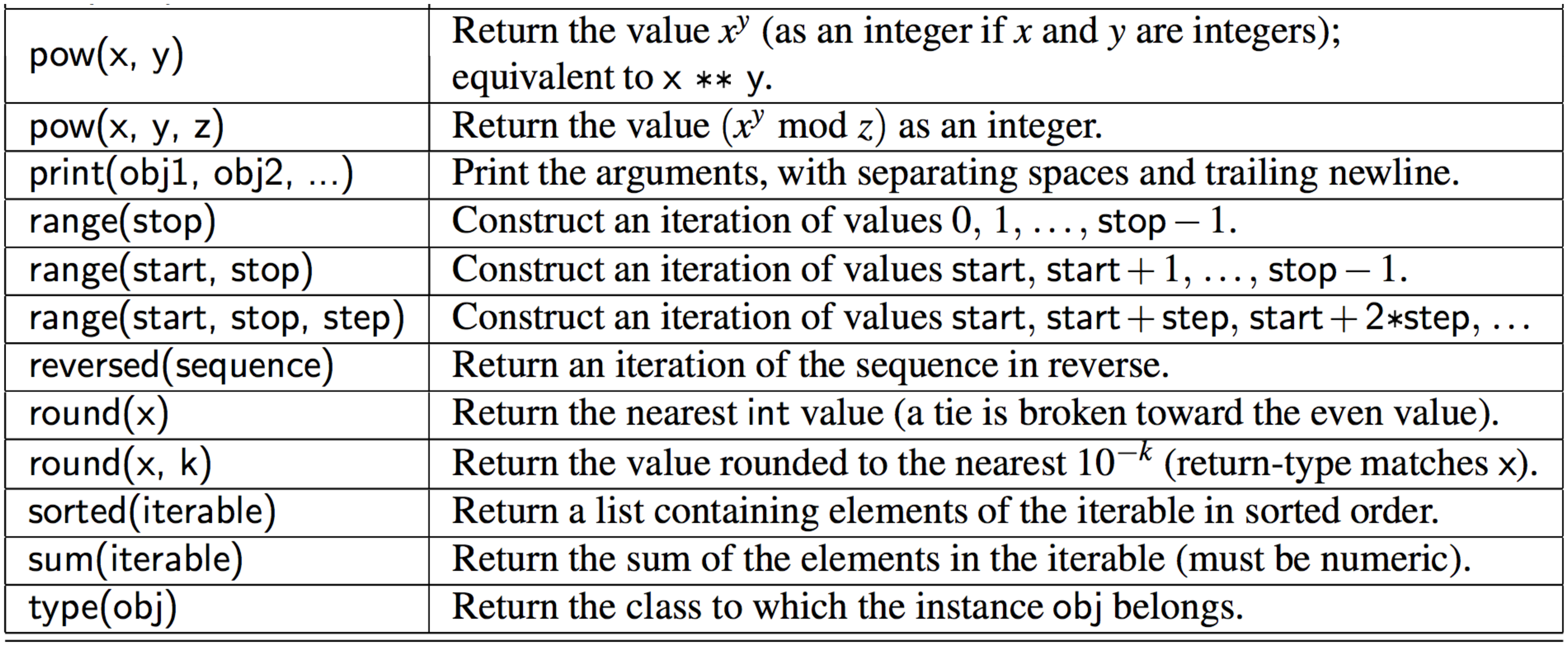
Built-in Functions (cont'd)



isinstance(obj, cls)	Determine if obj is an instance of the class (or a subclass).
iter(iterable)	Return a new iterator object for the parameter (see Section 1.8).
len(iterable)	Return the number of elements in the given iteration.
map(f, iter1, iter2, ...)	Return an iterator yielding the result of function calls f(e1, e2, ...) for respective elements e1 ∈ iter1, e2 ∈ iter2, ...
max(iterable)	Return the largest element of the given iteration.
max(a, b, c, ...)	Return the largest of the arguments.
min(iterable)	Return the smallest element of the given iteration.
min(a, b, c, ...)	Return the smallest of the arguments.
next(iterator)	Return the next element reported by the iterator (see Section 1.8).
open(filename, mode)	Open a file with the given name and access mode.
ord(char)	Return the Unicode code point of the given character.

```
c = map(operator.add,[-2,1,-9],[-1,-1,-1])
```

Built-in Functions (cont'd)



pow(x, y)	Return the value x^y (as an integer if x and y are integers); equivalent to x ** y.
pow(x, y, z)	Return the value $(x^y \bmod z)$ as an integer.
print(obj1, obj2, ...)	Print the arguments, with separating spaces and trailing newline.
range(stop)	Construct an iteration of values 0, 1, . . . , stop − 1.
range(start, stop)	Construct an iteration of values start, start + 1, . . . , stop − 1.
range(start, stop, step)	Construct an iteration of values start, start + step, start + 2*step, . . .
reversed(sequence)	Return an iteration of the sequence in reverse.
round(x)	Return the nearest int value (a tie is broken toward the even value).
round(x, k)	Return the value rounded to the nearest 10^{-k} (return-type matches x).
sorted(iterable)	Return a list containing elements of the iterable in sorted order.
sum(iterable)	Return the sum of the elements in the iterable (must be numeric).
type(obj)	Return the class to which the instance obj belongs.

Input & Output

- Output
  print(a, b, c, sep=‘:’); print(a,b,c,sep=“”)? print(..., end = ’#’)
  can also print to a file rather than console
- Input
  a prompt message for input, returns a string

```
year = int(input('In what year were you born? '))

reply = input('Enter x and y, separated by spaces: ')
pieces = reply.split( )     # returns a list of strings, as separated by spaces
x = float(pieces[0])
y = float(pieces[1])
```

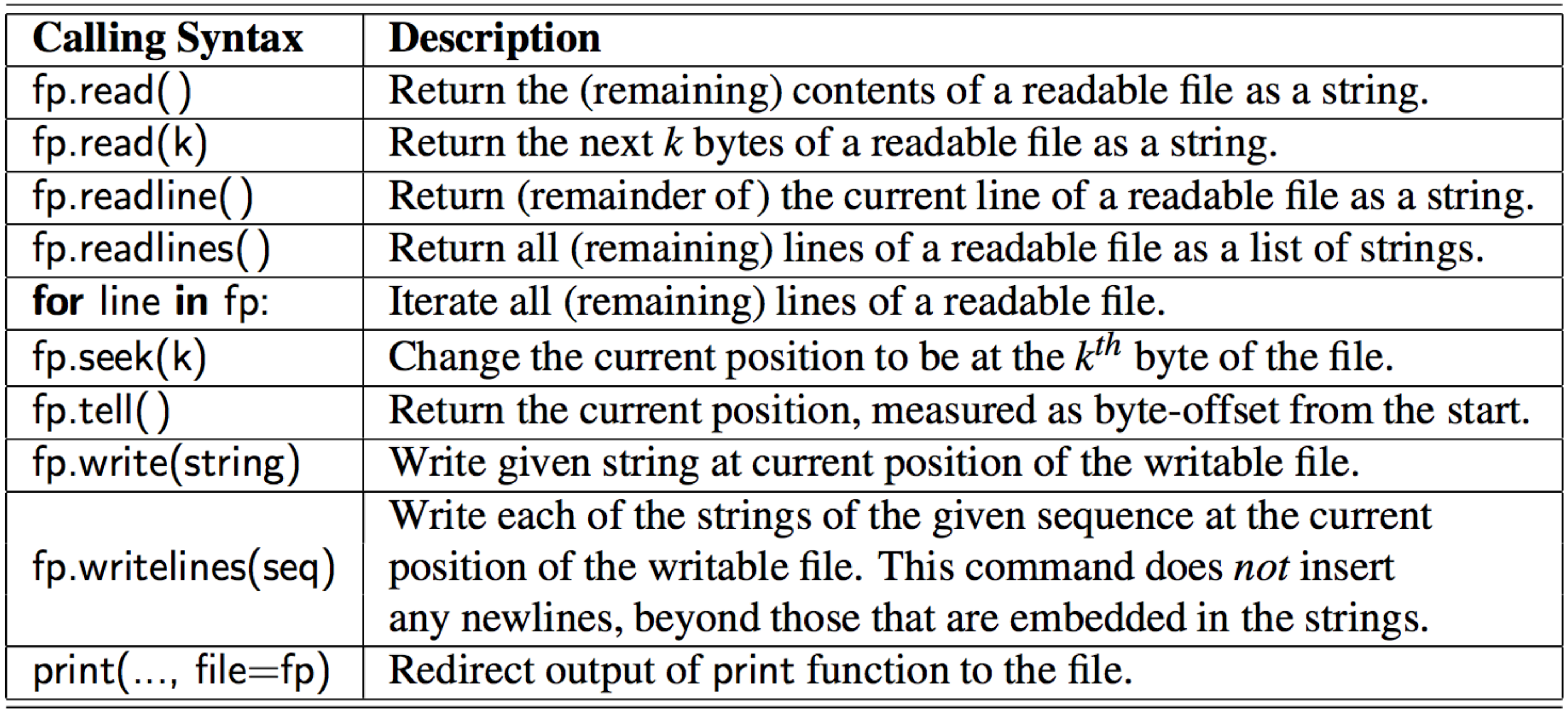
Input & Output (cont’d)

```
age = int(input('Enter your age in years: '))
max_heart_rate = 206.9 - (0.67 * age)    # as per Med Sci Sports Exerc.
target = 0.65 * max_heart_rate
print('Your target fat-burning heart rate is', target)
```

Files

- fp = open(‘sample.txt’); fp = open(‘sample.txt’, ‘w’);
  - default mode: ‘r’ – read only,
  - ‘w’ – writing, rewrite/create the whole thing
  - ‘a’ – appending to the end of the existing file
  - ‘wb’, ‘rb’ – treating the file as a binary file (for better storage efficiency)
- read/write (next slide)
- When done: fp.close()

Files



Calling Syntax	Description
fp.read( )	Return the (remaining) contents of a readable file as a string.
fp.read(k)	Return the next *k* bytes of a readable file as a string.
fp.readline( )	Return (remainder of) the current line of a readable file as a string.
fp.readlines( )	Return all (remaining) lines of a readable file as a list of strings.
for line **in** fp:	Iterate all (remaining) lines of a readable file.
fp.seek(k)	Change the current position to be at the k^{th} byte of the file.
fp.tell( )	Return the current position, measured as byte-offset from the start.
fp.write(string)	Write given string at current position of the writable file.
fp.writelines(seq)	Write each of the strings of the given sequence at the current position of the writable file. This command does *not* insert any newlines, beyond those that are embedded in the strings.
print(..., file=fp)	Redirect output of print function to the file.

- **for** line **in** fp:
- fp.write(‘Hello World.\n’) – write does not add end-of-line

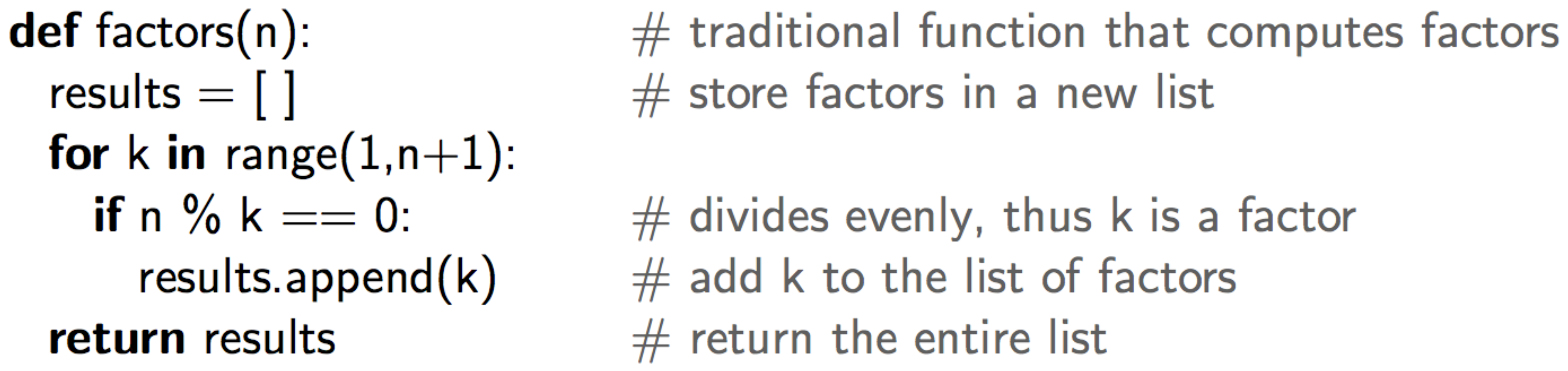
Iterator

- An ***iterator*** is an object that manages an iteration through a series of values.
- An ***iterable*** is an object, obj, that produces an *iterator* via the syntax iter(obj).
- data = [1, 2, 4, 8]; i = iter(data);
- Each call next(i) returns an element of data, until StopIteration exception
- seversed(s)
- Can have multiple ones. Will report updated values if the list is updated.
- More in future lectures

- Implicit iterables:
  range(1000000), lazy evaluation,
  to use: **for** j **in** range(1000000):
  Convert to list: list(range(1000000))

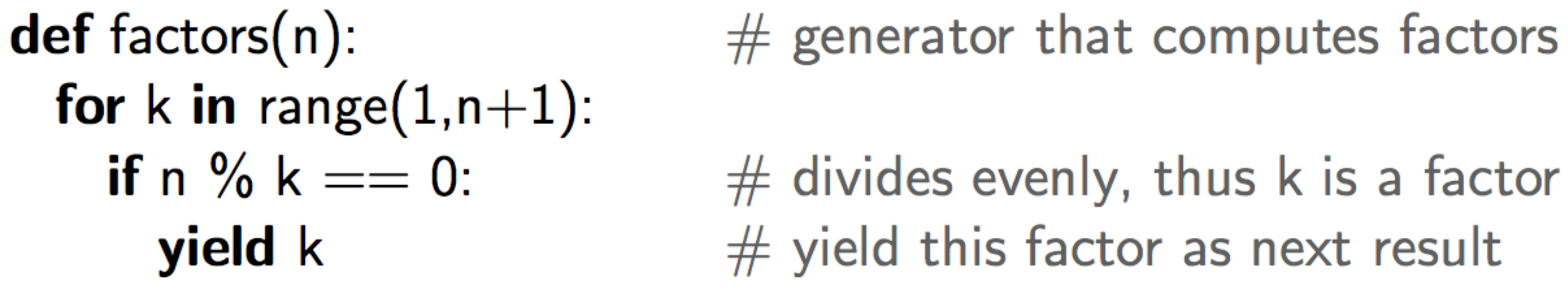
Generator

- Generator: a class acting like an iterator
- Implementation: use yield, not return
- A traditional function, returning a list of factors of n



```
def factors(n):                  # traditional function that computes factors
  results = [ ]                  # store factors in a new list
  for k in range(1,n+1):
    if n % k == 0:               # divides evenly, thus k is a factor
      results.append(k)          # add k to the list of factors
  return results                 # return the entire list
```

- A generator (usage: **for** factor **in** factors(100):)



```
def factors(n):                  # generator that computes factors
  for k in range(1,n+1):
    if n % k == 0:               # divides evenly, thus k is a factor
      yield k                    # yield this factor as next result
```

To Speed up

```
def factors(n):              # generator that computes factors
  k = 1
  while k * k < n:           # while k < sqrt(n)
    if n % k == 0:
      yield k
      yield n // k
    k += 1
  if k * k == n:             # special case if n is perfect square
    yield k
```

Compare at n = 100, what is the difference?

```
def factors(n):              # generator that computes factors
  for k in range(1,n+1):
    if n % k == 0:           # divides evenly, thus k is a factor
      yield k                # yield this factor as next result
```

Another Example

```
def fibonacci( ):
  a = 0
  b = 1
  while True:              # keep going...
    yield a                # report value, a, during this pass
    future = a + b
    a = b                  # this will be next value reported
    b = future             # and subsequently this
```

0, 1, 1, 2, 3, 5, 8, ….

Conditional Expression

expr1 **if** *condition* **else** *expr2*

- Improves readability (but do not abuse......)

```
if n >= 0:
    param = n
else:
    param = -n
result = foo(param)
```

```
param = n if n >= 0 else -n
result = foo(param)
```

- Even simpler: `result = foo(n if n >= 0 else -n)`

Packing / Unpacking of Sequences

- Automatic packing: comma-separated expressions --> a single tuple
  data = 2, 4, 6, 8 → data = (2, 4, 6, 8)
- Consider a function with return x,y
  - r = foo(...) → r is a tuple with (x,y)
  - rx, ry = foo(...) → rx = x, ry = y
- Examples:
  - quotient, remainder = divmod(a, b), or myPair = divmod(a,b)
  - a, b, c, d = range(7, 11)
  - **for** x, y **in** [(7, 2), (5, 8), (6, 4)]:
  - **for** k, v **in** my_dict.items():
- Simultaneous assignments:
  - x, y, z = 6, 2, 5
  - j, k = k, j
    # a convenient way to swap contents of j and k

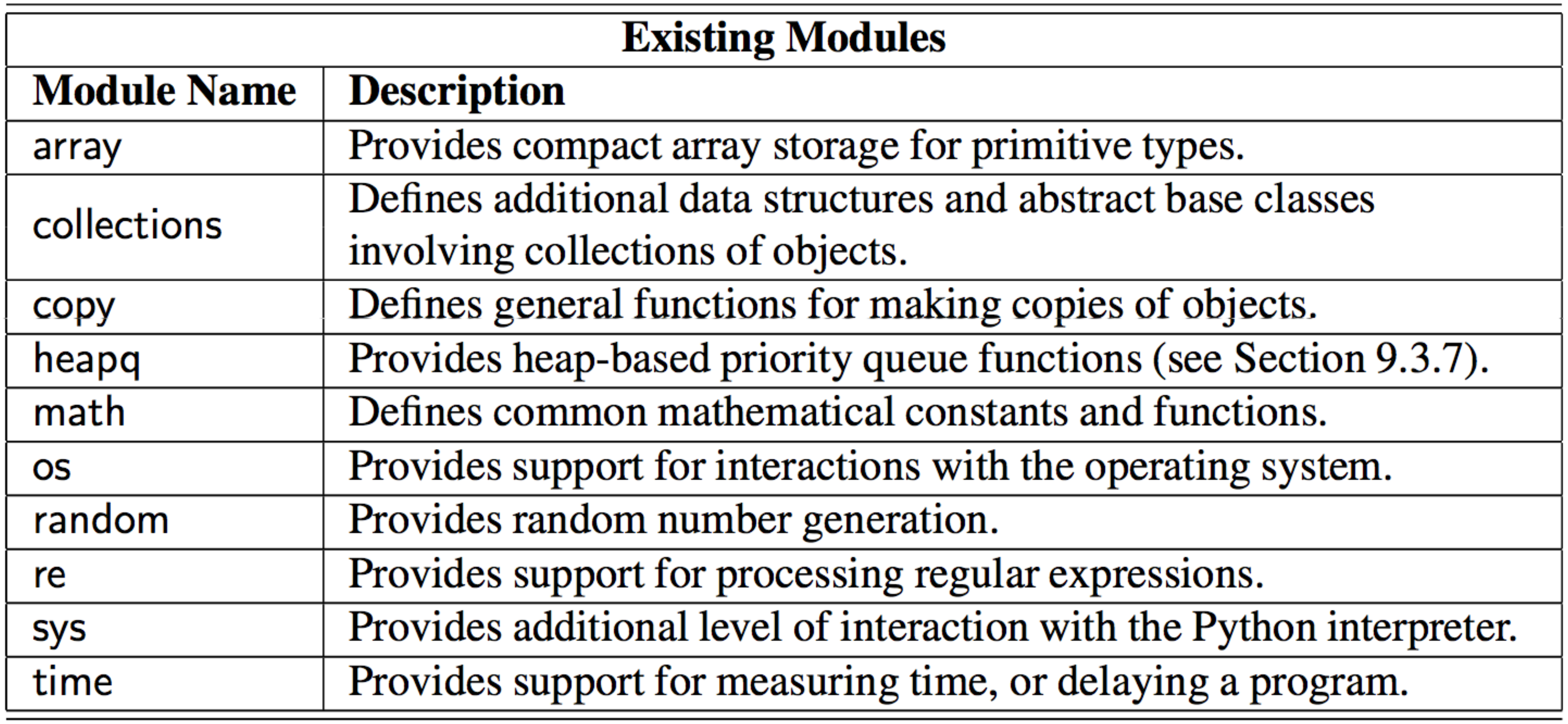
```
temp = j
j = k
k = temp
```

```
def fibonacci( ):
  a, b = 0, 1
  while True:
    yield a
    a, b = b, a+b
```

Modules and Import

- Additional libraries: **modules.** To use: **import**
- **from** math **import** pi, sqrt
  a = sqrt(pi)
- **import** math
  a = math.sqrt(math.pi)
- Aliasing for convenience
  import numpy **as** np
- Define count function in utility.py
  from utility **import** count (will explain soon)

Modules and Import



Existing Modules	
Module Name	**Description**
array	Provides compact array storage for primitive types.
collections	Defines additional data structures and abstract base classes involving collections of objects.
copy	Defines general functions for making copies of objects.
heapq	Provides heap-based priority queue functions (see Section 9.3.7).
math	Defines common mathematical constants and functions.
os	Provides support for interactions with the operating system.
random	Provides random number generation.
re	Provides support for processing regular expressions.
sys	Provides additional level of interaction with the Python interpreter.
time	Provides support for measuring time, or delaying a program.

Table 1.7: Some existing Python modules relevant to data structures and algorithms.

The end