CCPS109 Computer Science I L4

Lecturer: Nhan Tran

n3tran@Ryerson.ca

Agenda

Announcement

Midterm Feb 26@ 18:30

Lecture:

- Compound data types
  - Lists
  - tuples
- module/ codes in different files

Tuple

- an ordered sequence of elements, can mix element types
- cannot change element values, **immutable**
- Use with parentheses ()

```
te = ()                         #empty tuple

t = (7,"Hi",23)

t[0]                               #7

t2=(7,"Hi",23)+ (5,6)   # t2=(7,"Hi",23,5,6)
```

`t[1:2]` → slice tuple, evaluates to `("Hi",)`

`t[1:3]` → slice tuple, evaluates to `("Hi",23)`

`len(t)` → evaluates to 3

Tuple assignment

```
b = ("Bob", 19, "CS")      # tuple packing

(name, age, studies) = b    # tuple unpacking

print(name)                # 'Bob'

print(age)                 #  19

print (studies)            # 'CS'
```

Tuple swap

- conveniently used to **swap** variable values

```
x = y
y = x
```

✗

```
temp = x
x = y
y = temp
```

✓

```
(x, y) = (y, x)
```

✓

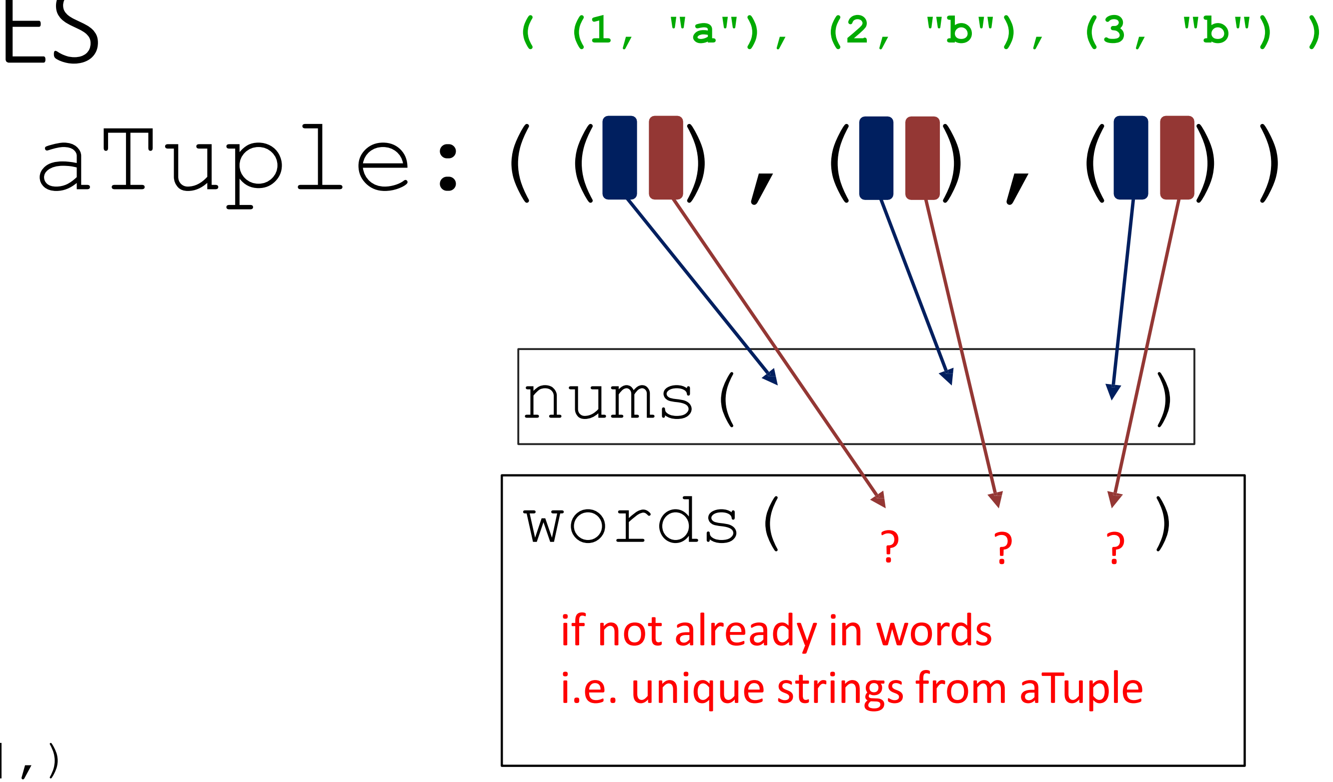
left side is a tuple of variables;
the right side is a tuple of values. Each value is assigned to its respective variable.
All the expressions on the right side are evaluated before any of the assignments
tuple assignment quite versatile.

Tuple

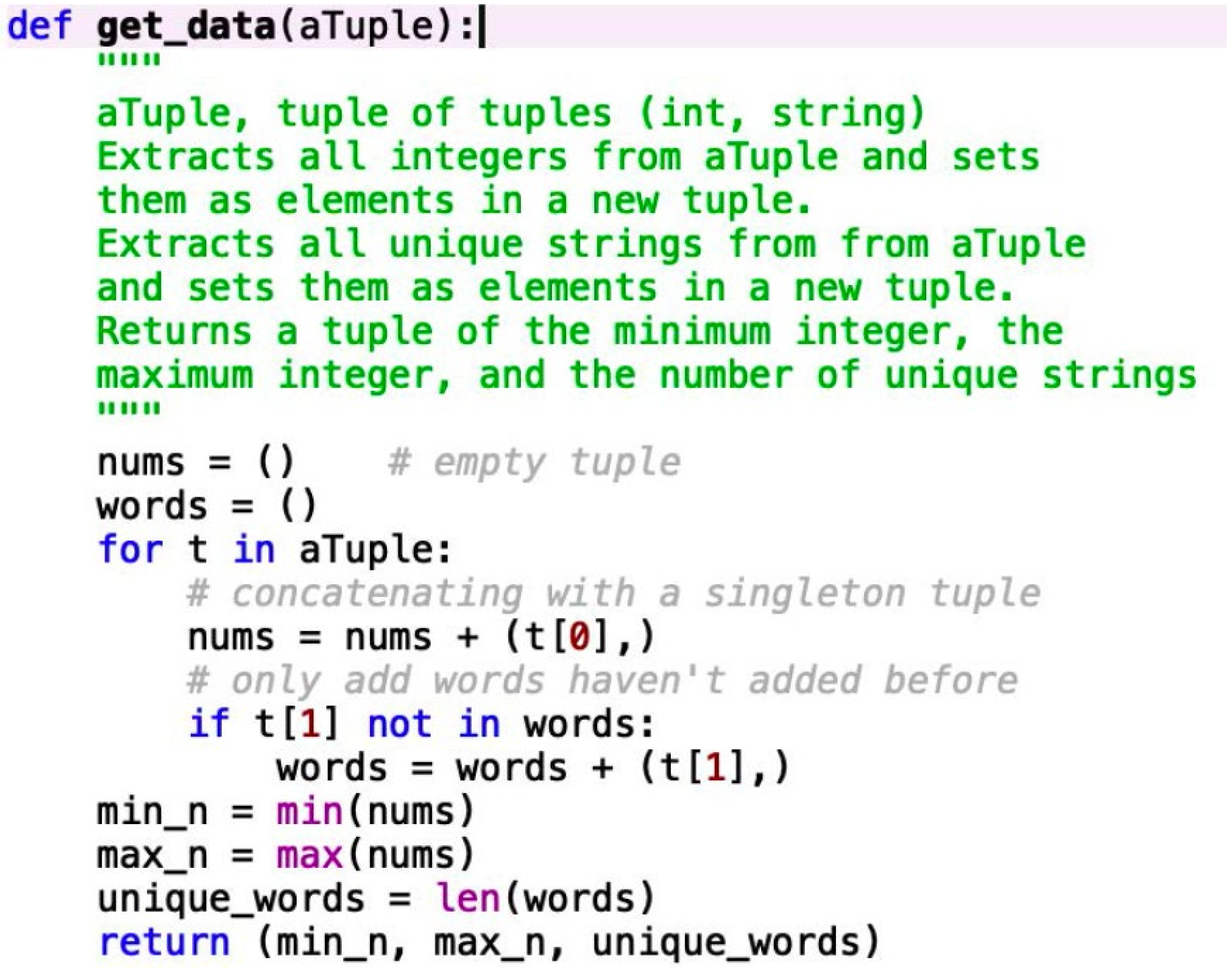
- used to **return more than one value** from a function

```
def quotient_and_remainder(x, y):   q = x // y
    r = x % y
    return (q, r)

(quot, rem) = quotient_and_remainder(4,5)
```

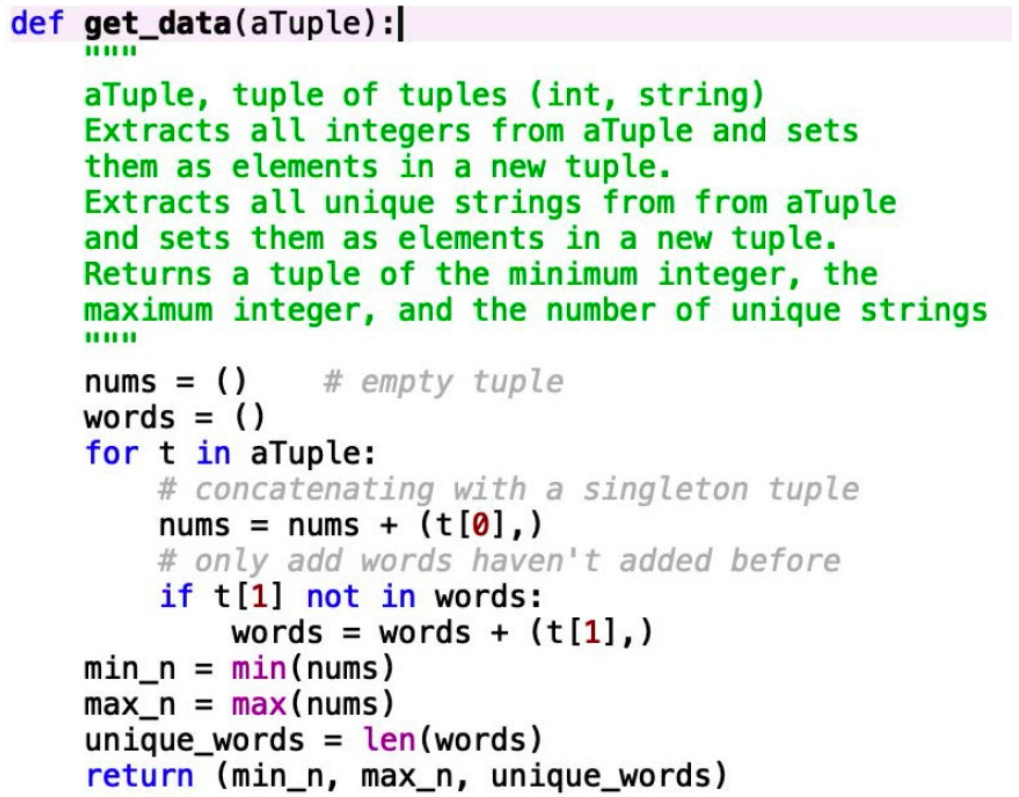
MANIPULATING TUPLES


((1, "a"), (2, "b"), (3, "b"))
aTuple: (( ), ( ), ( ))
nums ( )
words (? ? ?)
if not already in words
i.e. unique strings from aTuple

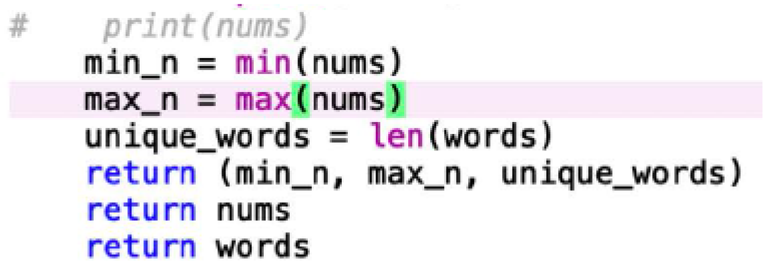
```
def get_data(aTuple):
    nums = ()
    words = ()
    for t in aTuple:
        nums = nums + (t[0],)
        if t[1] not in words:
            words = words + (t[1],)
    min_n = min(nums)
    max_n = max(nums)
    unique_words = len(words)
    return (min_n, max_n, unique_words)
```



```python
def get_data(aTuple):
    """
    aTuple, tuple of tuples (int, string)
    Extracts all integers from aTuple and sets
    them as elements in a new tuple.
    Extracts all unique strings from from aTuple
    and sets them as elements in a new tuple.
    Returns a tuple of the minimum integer, the
    maximum integer, and the number of unique strings
    """
    nums = ()    # empty tuple
    words = ()
    for t in aTuple:
        # concatenating with a singleton tuple
        nums = nums + (t[0],)
        # only add words haven't added before
        if t[1] not in words:
            words = words + (t[1],)
    min_n = min(nums)
    max_n = max(nums)
    unique_words = len(words)
    return (min_n, max_n, unique_words)
```

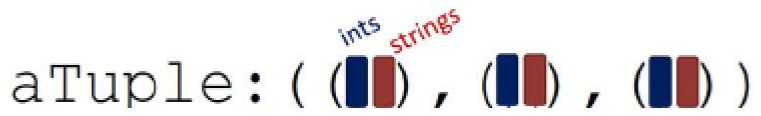


```python
def get_data(aTuple):
    """
    aTuple, tuple of tuples (int, string)
    Extracts all integers from aTuple and sets
    them as elements in a new tuple.
    Extracts all unique strings from from aTuple
    and sets them as elements in a new tuple.
    Returns a tuple of the minimum integer, the
    maximum integer, and the number of unique strings
    """
    nums = ()    # empty tuple
    words = ()
    for t in aTuple:
        # concatenating with a singleton tuple
        nums = nums + (t[0],)
        # only add words haven't added before
        if t[1] not in words:
            words = words + (t[1],)
    min_n = min(nums)
    max_n = max(nums)
    unique_words = len(words)
    return (min_n, max_n, unique_words)
```

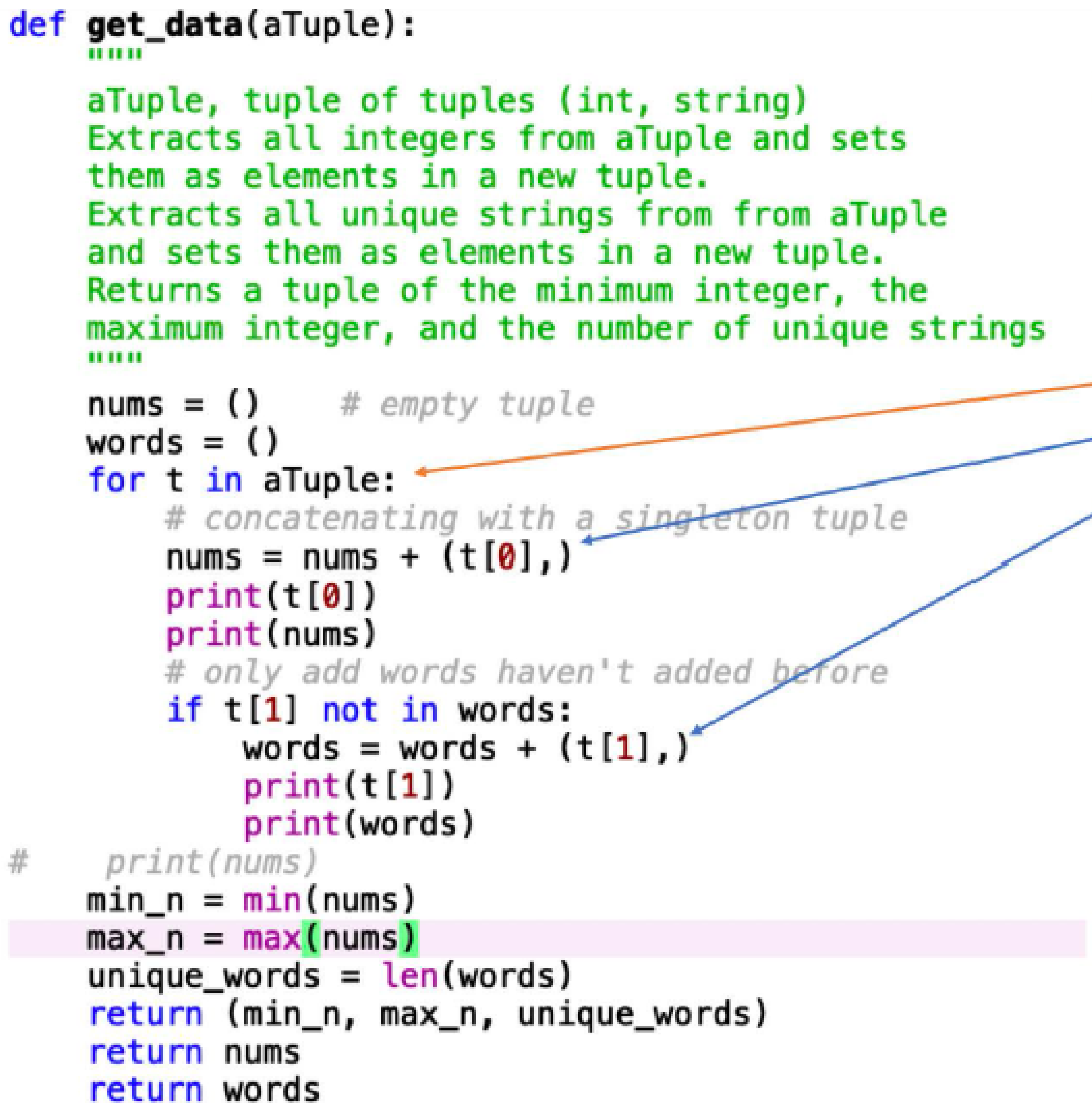


```python
#    print(nums)
    min_n = min(nums)
    max_n = max(nums)
    unique_words = len(words)
    return (min_n, max_n, unique_words)
    return nums
    return words
```

aTuple = ((1, “a”), (2, “b”), (3, ”a”), (4, “b”))



1. aTuple is a tuple where its elements are tuples
2. Extract all integers as a tuple
3. Extract all unique strings as a tuple
4. Find the minimum integer
5. Find the maximum integer
6. Find the number of unique strings
7. Return a tuple of the above 4, 5 and 6
8. Return nums and words for check



```python
def get_data(aTuple):
    """
    aTuple, tuple of tuples (int, string)
    Extracts all integers from aTuple and sets
    them as elements in a new tuple.
    Extracts all unique strings from from aTuple
    and sets them as elements in a new tuple.
    Returns a tuple of the minimum integer, the
    maximum integer, and the number of unique strings
    """
    nums = ()    # empty tuple
    words = ()
    for t in aTuple:
        # concatenating with a singleton tuple
        nums = nums + (t[0],)
        print(t[0])
        print(nums)
        # only add words haven't added before
        if t[1] not in words:
            words = words + (t[1],)
            print(t[1])
            print(words)
#    print(nums)
    min_n = min(nums)
    max_n = max(nums)
    unique_words = len(words)
    return (min_n, max_n, unique_words)
    return nums
    return words
```

aTuple = ((1, “a”), (2, “b”), (3, ”a”), (4, “b”))

For the 1st iteration:

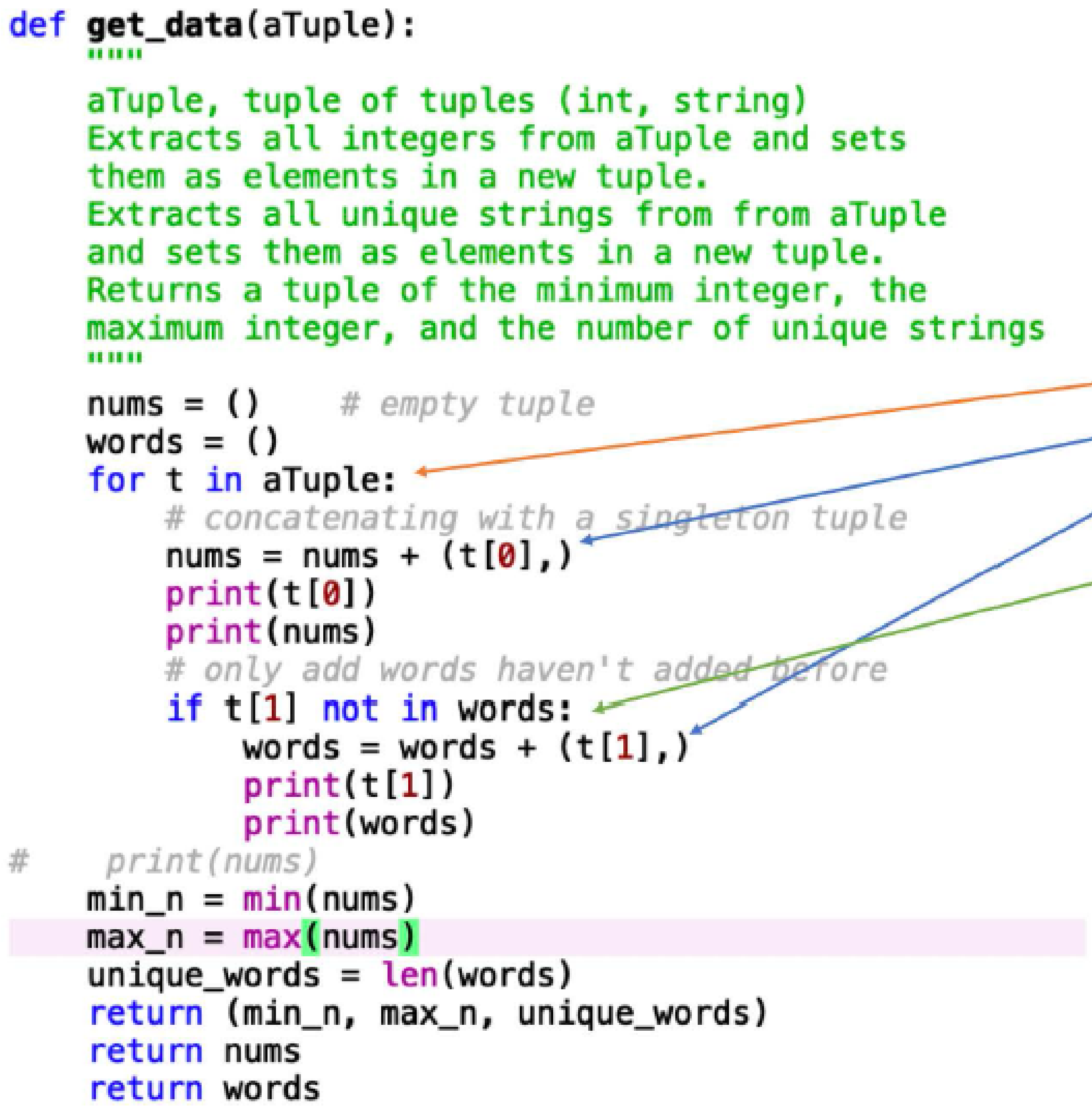
t is the first element of aTuple

t is a tuple

t = (1, “a”)

t[0] = 1

t[1] = ‘a’

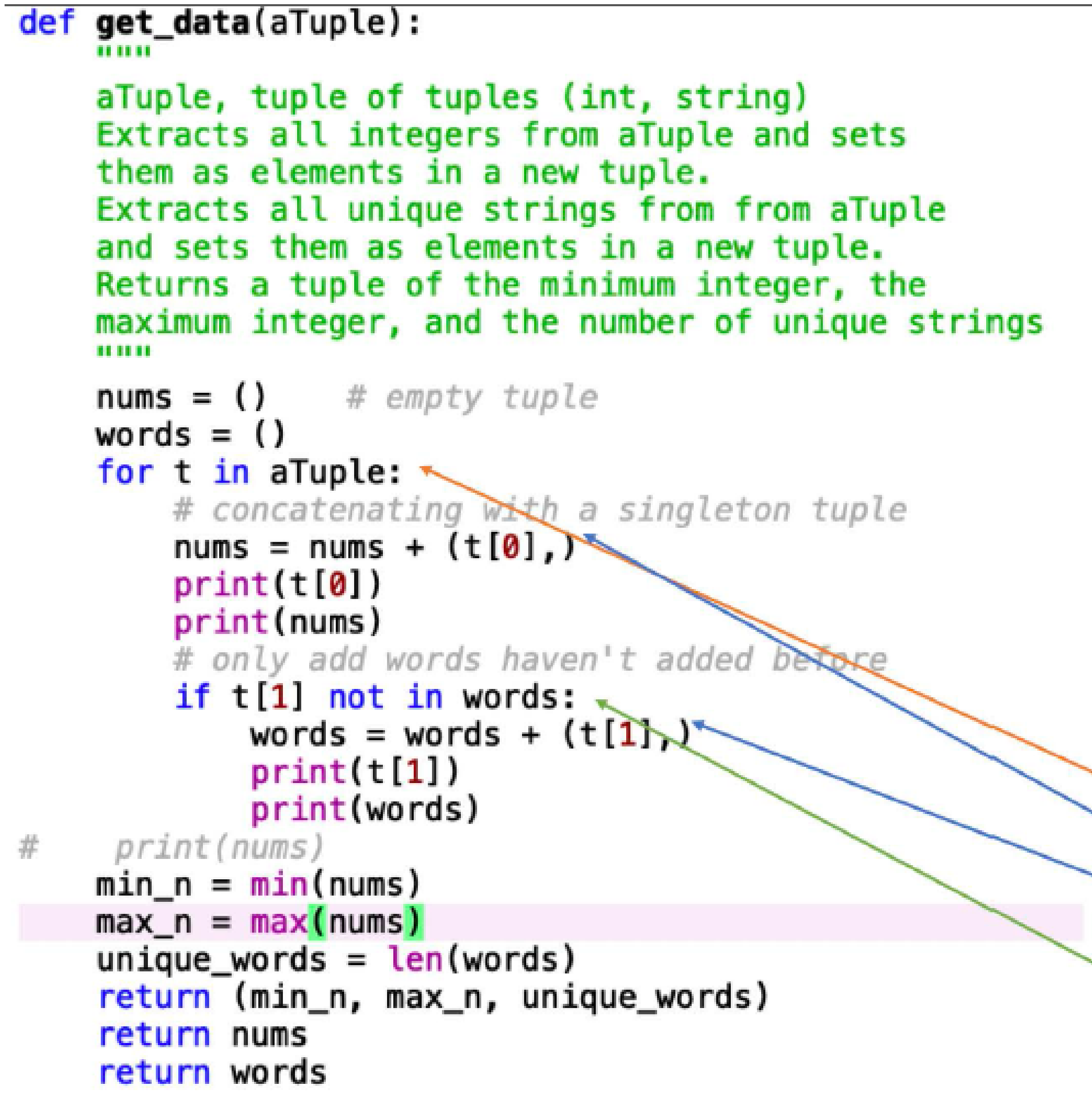


```
def get_data(aTuple):
    """
    aTuple, tuple of tuples (int, string)
    Extracts all integers from aTuple and sets
    them as elements in a new tuple.
    Extracts all unique strings from from aTuple
    and sets them as elements in a new tuple.
    Returns a tuple of the minimum integer, the
    maximum integer, and the number of unique strings
    """
    nums = ()    # empty tuple
    words = ()
    for t in aTuple:
        # concatenating with a singleton tuple
        nums = nums + (t[0],)
        print(t[0])
        print(nums)
        # only add words haven't added before
        if t[1] not in words:
            words = words + (t[1],)
            print(t[1])
            print(words)
#    print(nums)
    min_n = min(nums)
    max_n = max(nums)
    unique_words = len(words)
    return (min_n, max_n, unique_words)
    return nums
    return words
```

aTuple = ((1, “a”), (2, “b”), (3, ”a”), (4, “b”))

For the 1st iteration:

```
    # t is the first element of aTuple
    # t is a tuple
    t = (1, “a”)
    t[0] = 1
    t[1] = ‘a’
    nums = nums + (t[0],) = () + (1, ) = (1, )
    # if condition is satisfied
    words = words + (t[1], ) = () + (‘a’, ) = (‘a’, )
```



```python
def get_data(aTuple):
    """
    aTuple, tuple of tuples (int, string)
    Extracts all integers from aTuple and sets
    them as elements in a new tuple.
    Extracts all unique strings from from aTuple
    and sets them as elements in a new tuple.
    Returns a tuple of the minimum integer, the
    maximum integer, and the number of unique strings
    """
    nums = ()    # empty tuple
    words = ()
    for t in aTuple:
        # concatenating with a singleton tuple
        nums = nums + (t[0],)
        print(t[0])
        print(nums)
        # only add words haven't added before
        if t[1] not in words:
            words = words + (t[1],)
            print(t[1])
            print(words)
#    print(nums)
    min_n = min(nums)
    max_n = max(nums)
    unique_words = len(words)
    return (min_n, max_n, unique_words)
    return nums
    return words
```

aTuple = ((1, “a”), (2, “b”), (3, ”a”), (4, “b”))

For the 1st iteration:

# t is the first element of aTuple
# t is a tuple
t = (1, “a”)
t[0] = 1
t[1] = ‘a’
nums = nums + (t[0],) = () + (1,) = (1,)
# if condition is satisfied
words = words + (t[1],) = () + (‘a’,) = (‘a’,)

For the 2nd iteration:

# t is the second element of aTuple
t = (2, “b”)
t[0] = 2
t[1] = ‘b’
nums = nums + (t[0],) = (1,) + (2,) = (1, 2)
# if condition is satisfied
words = words + (t[1],) = (‘a’,) + (‘b’,) = (‘a’, ‘b’)

Tuple

- Cannot add elements to a tuple:
  - `no append() or extend() method`
- You can’t remove elements from a tuple:
  - `no remove() or pop() method`
- Can index, slice
- Use less memory than list
- slightly faster in indexing speed than list

List

Object stored data in ordered sequence and accessible by index

- list is denoted by square brackets, []
- a list contains elements
- Elements
  - Homogeneous
  - May contain mixed types
- list elements can be changed   mutable

Indices and order

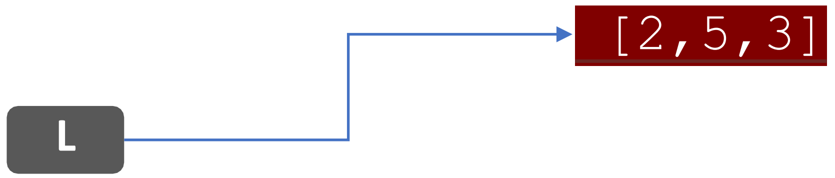
```
L= []                       # create an empty list
L = [2, 'a', 4, [1,2]]
len(L)                      #4
L[0]                        #2
L[2]+1                      #5     ←4+1
L[3]                        #[1,2]    another list
L[4]                        #error  : index out of range
i=2
L[i-1]                      # evaluate to ‘a’ since L[1] store ‘a’
```

Change Elements

- lists are **mutable**!
- assigning to a new element at an index changes the value

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

- `L` is now `[2, 5, 3]`, note this is the **same object** `L`



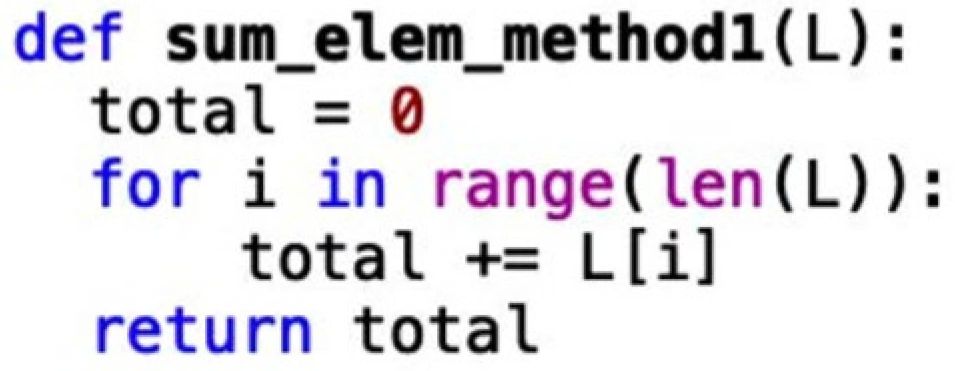
Iterating Over A List

- compute the **sum of elements** of a list
- common pattern, iterate over list elements

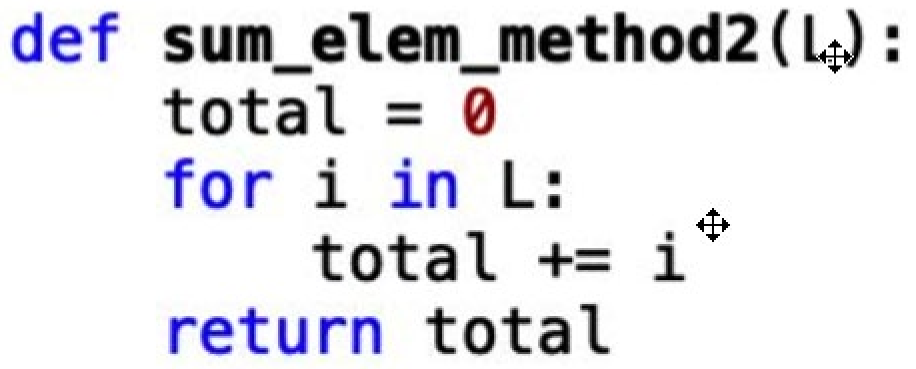
```
total = 0
  for i in range(len(L)):
      total += L[i]
  print total
```

- note:
  - list elements are indexed `0` to `len(L)-1`
  - `range(n)` goes from `0` to `n-1`

Iterating Over A List



```python
def sum_elem_method1(L):
  total = 0
  for i in range(len(L)):
      total += L[i]
  return total
```

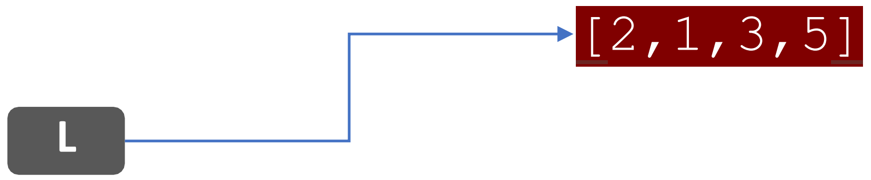


```python
def sum_elem_method2(L):
    total = 0
    for i in L:
        total += i
    return total
```

Lists Operations

- **add** elements **to end** of list with `L.append(element)`
- All elements are passed as a single element

```
L = [2,1,3]
L.append(5)
```

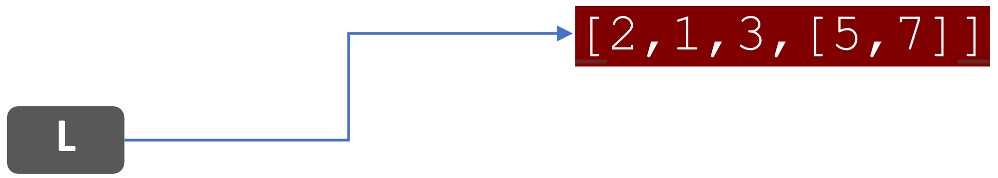


- Reminder
  - lists are Python objects, everything in Python is an object
  - objects have data
  - objects have methods and functions
  - access this information by `object_name.do_something()`
  - will learn more about these later

Lists Operations

- add elements to end of list with `L.append(element)`
- All elements are passed as a single element

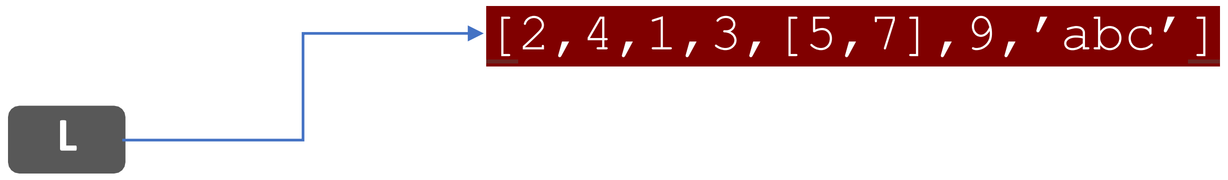
```
L = [2,1,3]
L.append([5,7])       #adding one element
```



Lists Operations

- extend() adds the elements one-by-one into the list

```
L = [2,1,3]
L.append([5,7])        #adding one element
L.extend([9,’abc’])
L.insert(1,4)
```

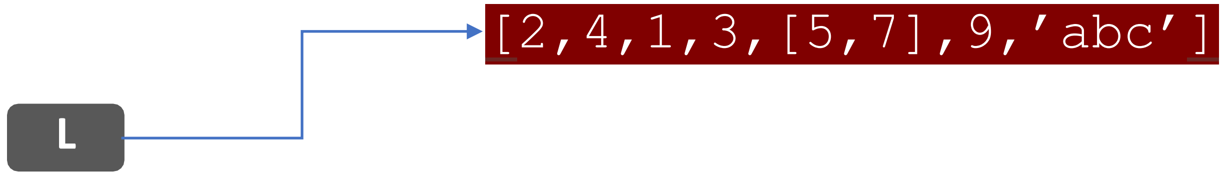


```
L1=[2,1,3]
L2=[4,5,6]
L3=L1 + L2              #concatenate L3 is [2,1,3,4,5,6]
L1.extend([0,6])        # L1 is [2,1,3,0,6]
```

Lists Operations

- extend() adds the elements one-by-one into the list

```
L = [2,1,3]
L.append([5,7])        #adding one element
L.extend([9,'abc'])
L.insert(1,4)
```



```
L1=[2,1,3]
L2=[4,5,6]
L3=L1 + L2                  #concatenate L3 is [2,1,3,4,5,6]
L1.extend([0,6])            # L1 is [2,1,3,0,6]
```

Lists Operations

- delete element at a **specific index** with `del(L[index])`
- remove a **specific element** with `L.remove(element)`
  - looks for the element and removes it
  - if element occurs multiple times, removes first occurrence
  - if element not in list, gives an error
- remove element at **end of list** with `L.pop()`, returns the removed element

```
L = [2,1,3,6,3,7,0] # do below in order
L.remove(2)  #L = [1,3,6,3,7,0]
L.remove(3)  #L = [1,6,3,7,0]
 del(L[1])    #L = [1,3,7,0]
 L.pop()                #L = [1,3,7]
 L.clear()       #L[]
```

List Operation

- index():finds the index value of value passed
  - if more than one match return the index of the first matched
- count(): count all number of times the passed value occurred.
- sorted() and sort() :sort the values of the list.
  - sorted() return a new sorted list   #accept list or any iterable

    ```
    mysorted=sorted([5,2,3,,4])
    ```

    - `May be use for tuple`

  - sort() modifies the original list (defined only for list)

    ```
    mylist.sort()
    ```

Additional reading on sort() key functions:
https://docs.python.org/3/howto/sorting.html#sortinghowto

List to string

- convert **string to list** with `list(s)`, returns a list with every character from `s` an element in `L`
- can use `s.split()`, to **split a string on a character** parameter, splits on spaces if called without a parameter
- use `''.join(L)` to turn a **list of characters into a string**, can give a character in quotes to add char between every element

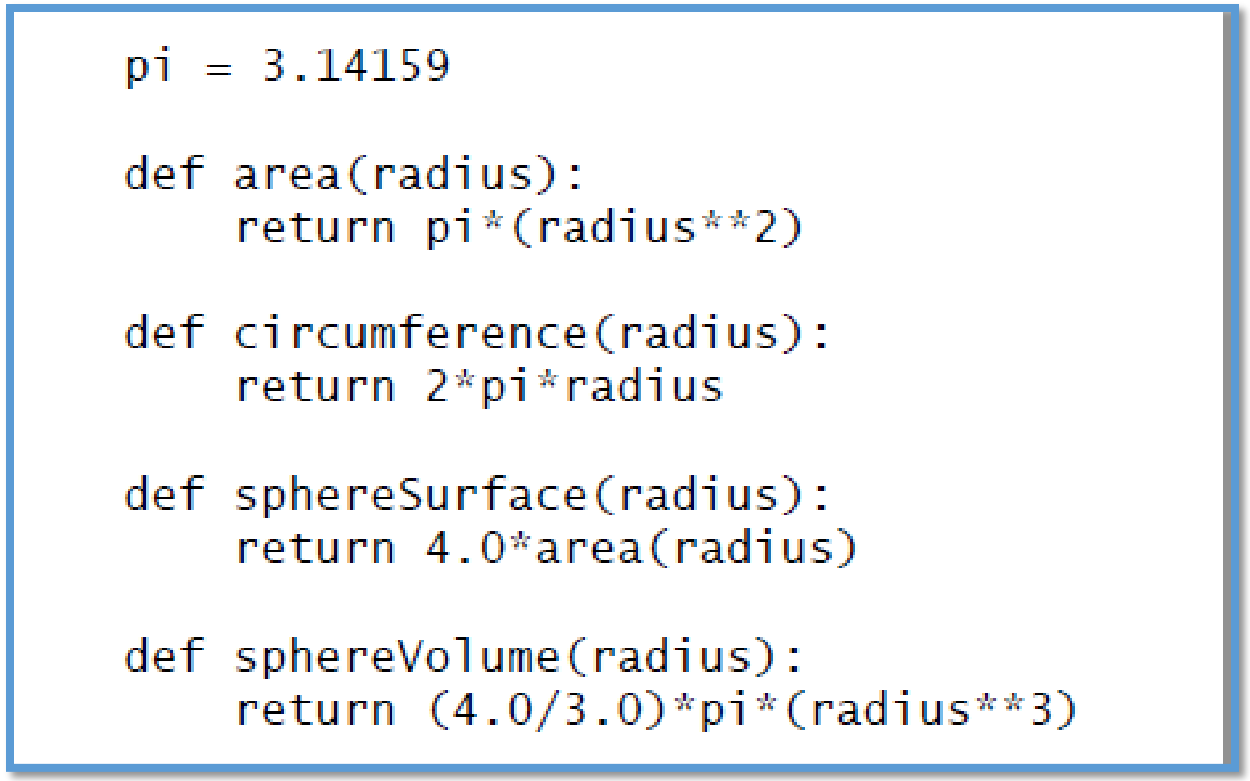
`s = "I<3 cs"`	→ `s` is a string
`list(s)`	→ returns `['I','<','3',' ','c','s']`
`s.split('<')`	→ returns `['I', '3 cs']`
`L = ['a','b','c']`	→ `L` is a list
`''.join(L)`	→ returns `"abc"`
`'_'.join(L)`	→ returns `"a_b_c"`

- `sort()` and `sorted()`
- `reverse()`
- and many more!
  https://docs.python.org/3/tutorial/datastructures.html

```
L=[9,6,0,3]
sorted(L)          → returns sorted list, does not mutate L
L.sort()
L.reverse()        → L=[9,6,3,0]
```

MODULES

- A module is a .py file containing Python definitions and statements.
  - For example, here is a file circle.py containing the codes as follows. We can name it as a module “circle”



```
pi = 3.14159

def area(radius):
    return pi*(radius**2)

def circumference(radius):
    return 2*pi*radius

def sphereSurface(radius):
    return 4.0*area(radius)

def sphereVolume(radius):
    return (4.0/3.0)*pi*(radius**3)
```

MODULES

- A file circle.py containing the codes.
- How to get access to a module?
- Using an import statement!

circle.py

Module: circle

```
pi = 3.14159

def area(radius):
    return pi*(radius**2)

def circumference(radius):
    return 2*pi*radius

def sphereSurface(radius):
    return 4.0*area(radius)

def sphereVolume(radius):
    return (4.0/3.0)*pi*(radius**3)
```

```
import circle
print(circle.pi)
print(circle.area(3))
print(circle.circumference(3))
print(circle.sphereSurface(3))
```

will print

```
3.14159
28.27431
18.84954
113.09724
```

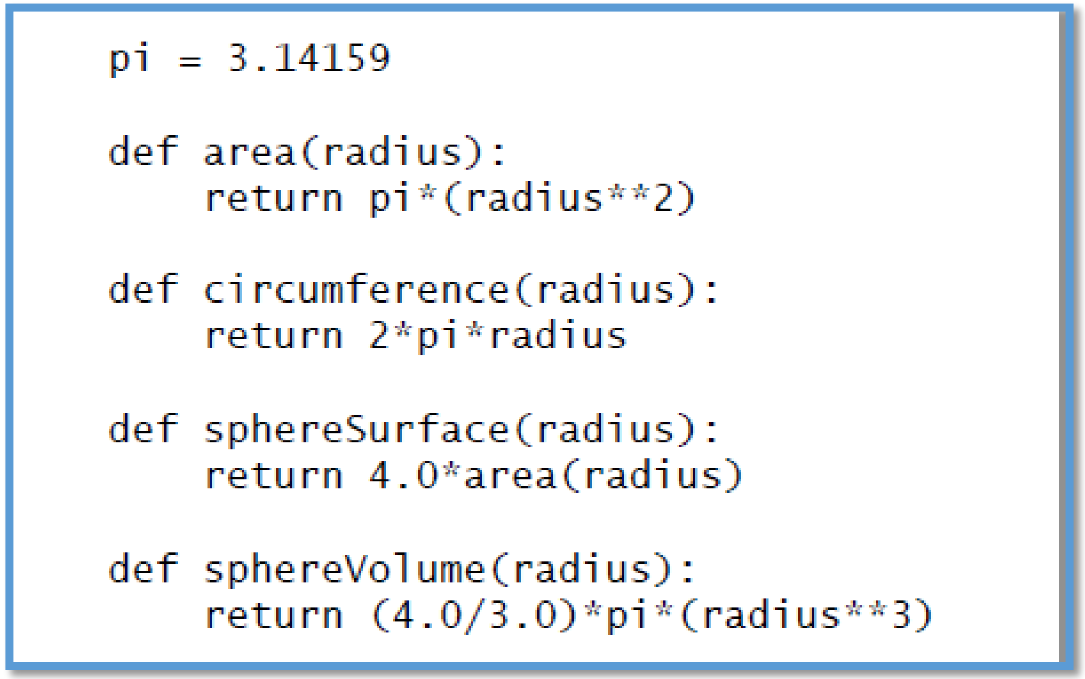
Module

- Module alias

```
import modulename as myAlias

import circle as mycircle
print(mycircle.sphereSurface(4))
```

- Import only what you needed

```
from modulename import specificfunction

from circle import area
print(area(3))
```



```
pi = 3.14159

def area(radius):
    return pi*(radius**2)

def circumference(radius):
    return 2*pi*radius

def sphereSurface(radius):
    return 4.0*area(radius)

def sphereVolume(radius):
    return (4.0/3.0)*pi*(radius**3)
```

Module *

- Import all except once beginning with an undercore (_)

```
from modulename import *

from circle import *
print(circumference(3))
```

- For additional info:
- https://docs.python.org/3/tutorial/modules.html

FILES

- File handle: access files

Create a file wit the name *kidsfile* and return a file handle for the *kids*file file

```
nameHandle=opend('kidsfile','w')
```

- Writing: the argument *‘w’* to open indicates that the file is to be opened for writing.

Create a file wit the name *kidsfile* and write the file by the input names line by line

```
nameHandle=opend('kidsfile','r')
for i in range(5):
        name=input('Please Enter Name: ')
        nameHandle.writ(name+'\n')
nameHandle.close()
```

FILES

- Reading: the argument *'r'* is to open the file to be reading.

Create a file with the name *kidsfile* and read the file line by line

```
nameHandle=opend('kidsfile','r')
for line in nameHandle:
      print(line)
nameHandle.close()
```

File Modes

r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Same as r but opens a file for both reading and writing
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist it creates a new file for writing.
w+	Same as w, but opens a file for both writing and reading.
a	Opens a file for appending. If the file exists- The file pointer is at the end of the file. If the file does not exist, it creates a new file for writing. The file opens in the append mode
a+	Same as a but opens a file for both appending and reading.

open(fn, 'w') fn is a string representing a file name. Creates a file for writing and returns a file handle.

open(fn, 'r') fn is a string representing a file name. Opens an existing file for reading and returns a file handle.

open(fn, 'a') fn is a string representing a file name. Opens an existing file for appending and returns a file handle.

fh.read() returns a string containing the contents of the file associated with the file handle fh.

fh.readline() returns the next line in the file associated with the file handle fh.

fh.readlines() returns a list each element of which is one line of the file associated with the file handle fh.

fh.write(s) write the string s to the end of the file associated with the file handle fh.

fh.writeLines(S) S is a sequence of strings. Writes each element of S to the file associated with the file handle fh.

fh.close() closes the file associated with the file handle fh.

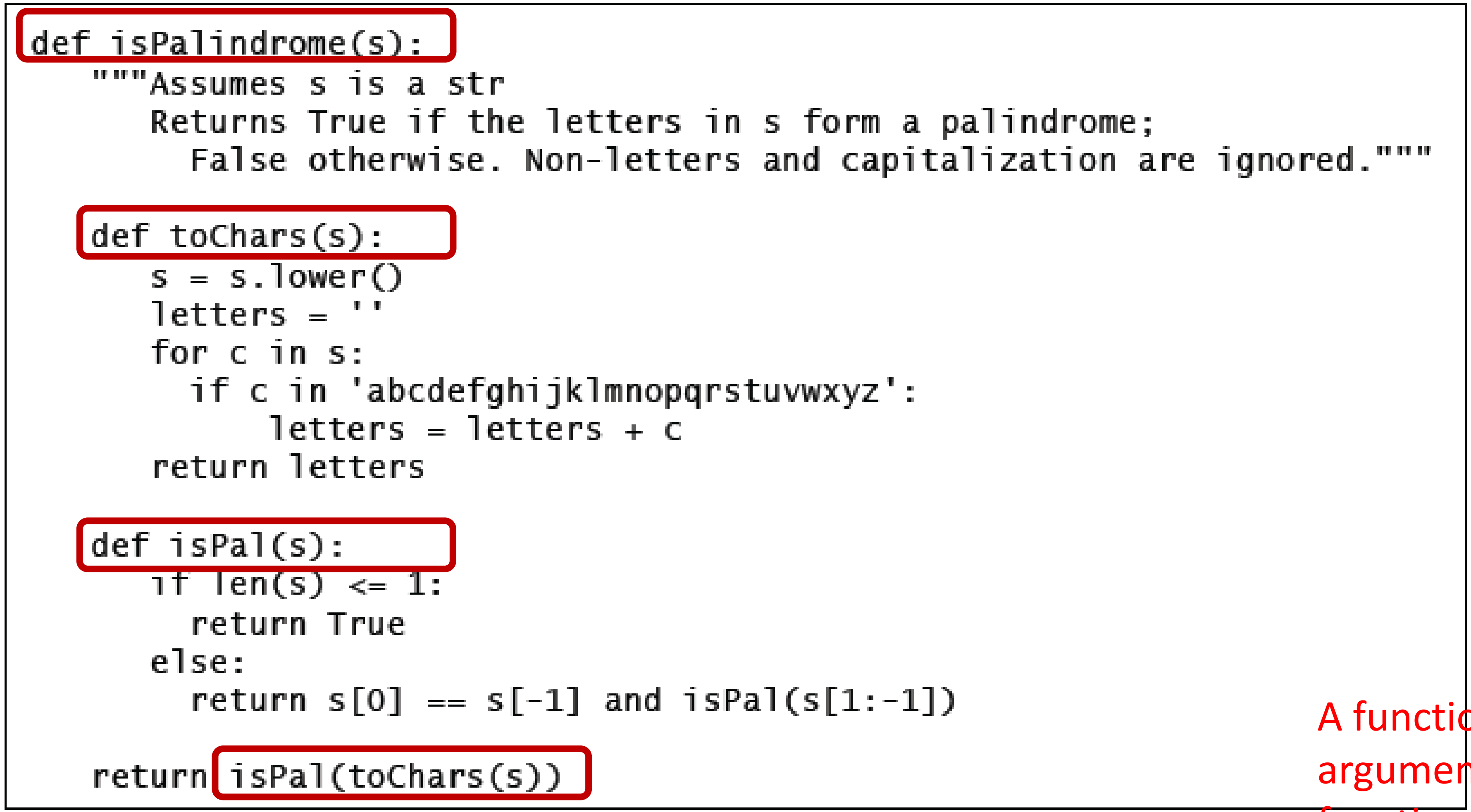
Figure 4.11 Common functions for accessing files

Palindrome

- The following is another version of the for determine whether a sequence is a palindrome

PALINDROMES

Palindromes: madam, nurses run



```
def isPalindrome(s):
   """Assumes s is a str
      Returns True if the letters in s form a palindrome;
        False otherwise. Non-letters and capitalization are ignored."""

   def toChars(s):
      s = s.lower()
      letters = ''
      for c in s:
        if c in 'abcdefghijklmnopqrstuvwxyz':
            letters = letters + c
      return letters

   def isPal(s):
      if len(s) <= 1:
        return True
      else:
        return s[0] == s[-1] and isPal(s[1:-1])

   return isPal(toChars(s))
```

Functions are inside a function!

A function is the argument of another function!

PALINDROMES

Recipes:

- Input: a string
- Output: return true or false to be a palindrome

1. Only keep lower-case letters, remove others;
2. Iteratively remove the first and the last letter if they are the same, until there is only a letter or there are only two repeated letters.
3. Return the output.

```
def isPalindrome(s):
   """Assumes s is a str
      Returns True if s is a palindrome; False otherwise.
        Punctuation marks, blanks, and capitalization are
        ignored."""

   def toChars(s):
      s = s.lower()
      letters = ''
      for c in s:
        if c in 'abcdefghijklmnopqrstuvwxyz':
            letters = letters + c
      return letters

   def isPal(s):
      print '  isPal called with', s
      if len(s) <= 1:
         print '  About to return True from base case'
         return True
      else:
         answer = s[0] == s[-1] and isPal(s[1:-1])
         print '  About to return', answer, 'for', s
         return answer

   return isPal(toChars(s))

def testIsPalindrome():
   print 'Try dogGod'
   print isPalindrome('dogGod')
   print 'Try doGood'
   print isPalindrome('doGood')
```

PALINDROMES

Recipes:

- Input: a string
- Output: return true or false to be a palindrome

1. Only keep lower-case letters, remove others;
2. Iteratively remove the first and the last letter if they are the same, until there is only a letter or there are only two repeated letters.
3. Return the output.

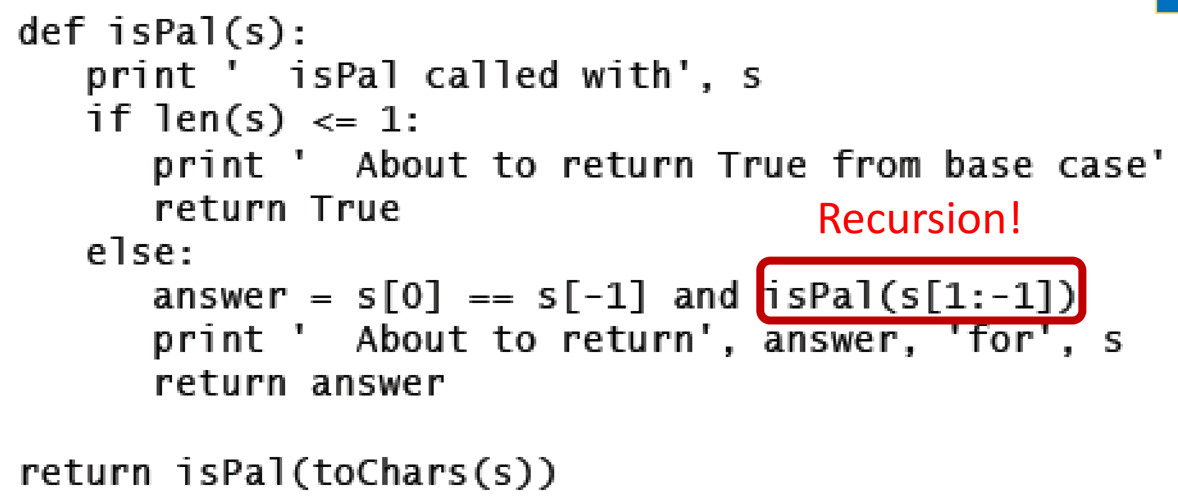
```
def isPalindrome(s):
   """Assumes s is a str
      Returns True if s is a palindrome; False otherwise.
        Punctuation marks, blanks, and capitalization are
        ignored."""

   def toChars(s):
      s = s.lower()
      letters = ''
      for c in s:
        if c in 'abcdefghijklmnopqrstuvwxyz':
            letters = letters + c
      return letters

   def isPal(s):
      print '  isPal called with', s
      if len(s) <= 1:
         print '  About to return True from base case'
         return True
      else:
         answer = s[0] == s[-1] and isPal(s[1:-1])
         print '  About to return', answer, 'for', s
         return answer

   return isPal(toChars(s))

def testIsPalindrome():
   print 'Try dogGod'
   print isPalindrome('dogGod')
   print 'Try doGood'
   print isPalindrome('doGood')
```

PALINDROMES

Recipes for isPal(s):

- Input: a letter-only and lower-case string
- Output: return true or false to be a palindrome

1. Check the forward direction of the string “s” if the first and the last letters are the same;
2. Recursion: Remove the first and last letters form the “s” and check the substring of the string “s” to repeat step 1;
3. Exit until the base cases (only a letter or two same letters are left) are satisfied.



```
def isPal(s):
    print '  isPal called with', s
    if len(s) <= 1:
        print '  About to return True from base case'
        return True
    else:
        answer = s[0] == s[-1] and isPal(s[1:-1])
        print '  About to return', answer, 'for', s
        return answer

return isPal(toChars(s))
```

Recursion!

madam
ada
d

doggod
oggo
gg