Computer Science Principles

CHAPTER 3 – ITERATION, LISTS, AND ALGORITHM DESIGN

1

Announcements

Reading: Read Chapter 3 of Conery

First quiz on Monday

Acknowledgement: These slides are revised versions of slides prepared by Prof. Arthur Lee, Tony Mione, Alex Kuhn and Pravin Pawar for earlier CSE 101 classes. Some slides are based on Prof. Kevin McDonald at SBU CSE 101 lecture notes and the textbook by John Conery.

2

Overview

This lecture will focus on:

i. **iteration** (code that repeats a list of steps)
ii. **lists**
iii. the thought process for **designing algorithms**

As an example, we will look at the ancient algorithm for finding prime numbers: **the Sieve of Eratosthenes**

3

Prime Numbers

A **prime** is a natural number greater than 1 that has no divisors other than 1 and itself

Non-prime numbers are called composite numbers

Example primes: 2, 3, 5, 11, 73, 9967, . . .

Example composites: 4 (2x2), 10 (2x5), 99 (3x3x11)

Prime numbers play an important role in encrypting data and Internet traffic

4

The Sieve of Eratosthenes

The basic idea of the algorithm is simple. Below, it is briefly described in pseudocode:

make a list of numbers, starting with 2
repeat the following steps until done:
the first unmarked number in the list is prime
cross off multiples of the most recent prime

So, first cross off multiples of 2.

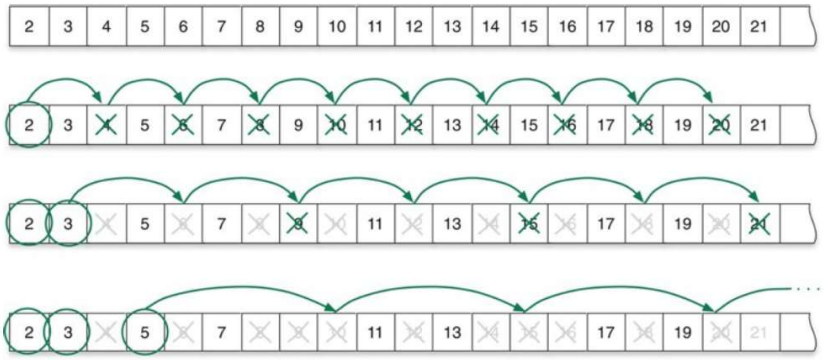
Then, cross off multiples of 3 that were not crossed off in the first round
- e.g., 6 is a multiple of 2 and 3, so it was crossed off in the first round

Next, cross off multiples of 5 that were not crossed off in the first two rounds
- Note that because 4 is a multiple of 2, all multiples of 4 were crossed off in the first round

5

The Sieve of Eratosthenes

The algorithm continues in this fashion until there are no more numbers to cross off



We will discuss more later exactly when it stops running

6

Devising an algorithm

The method depicted in the previous slide works well for short lists

But what if prime numbers between 2 and 100 are needed? …or 1000?

- It's a tedious process to write out a list of 100 numbers
- Chances are a few arithmetic mistakes will be made (this is a boring job!)

Can this method be turned into a computation?

Yes, but we need to add more detail to the steps

7

Devising an algorithm

A detailed specification of the starting condition is there in the pseudocode (e.g., "make a list")

However, some things are not clearly defined:

- "Cross off" and "next number" need to be clearly defined if this will be coded in Python
- The stopping condition is also not clear
  - When does the process stop? Perhaps when all the numbers are crossed off?

First, let us explore a few new ideas in Python

8

Collections

In everyday life, collections of objects are often encountered
- Course catalog: a collection of course descriptions
- Parking lot: a collection of vehicles

Mathematicians also work with collections
- Matrix (a table of numbers)
- Sequence (e.g., 1, 1, 2, 3, 5, 8, ...)

In computer science collections are made by defining a **data structure** that includes references to **objects**

The term object means *a piece of data*
- Objects include numbers, strings, dates, and more

9

Lists

An object that contains other objects is called a **container**

The simplest kind of container in Python is called a **list**

One way to make a list is to enclose a set of objects in square brackets:

```
ages = [61, 32, 19, 37, 42, 39]
```

The above statement is an assignment statement
- Python creates an object to represent the list and associates the name **ages** with the new object

The **len** function tells us how many elements are in a list:
- **len(ages) # returns the value 6**

10

Lists of strings

Any kind of object can be stored in a list

This statement defines a list with three strings:

- **breakfast = ['green eggs', 'ham', 'toast']**

Note what happens when we ask Python how many objects are in this list:

- **len(breakfast) # returns the value 3**
- The list contains three string objects, so the return value of the call to len is 3
- Python did not count the individual letters with a list

However, **len('apple')** returns 5 … with a string, it counts the individual letters

11

Empty lists

A list can also be made with no objects:

- **cars = []**

An empty list is still a list, even though it contains no objects

- A bag with nothing in it is still a bag, even though it contains nothing

The length of an empty list is 0

- **len(cars) # returns the value 0**

It may seem strange to create a list with nothing in it, but usually it is done because the list is needed but it will be filled later

12

Iteration

After building a container, most applications need to do something with each item in it

The idea is to "**step through**" the container to do something to each object

This type of operation is called **iteration**

For example, to find the largest item in an (unsorted) list, an algorithm would need to check the value of every item during its search

- This algorithm will be examined a little later

For loops

The simplest way to "visit" every item in a list is to use a **for** loop

This example prints every item in the list cars :

```
for car in cars:          # "for each car in a list of cars"
    print(car)
```

Note that the statements inside a for loop – the body of the loop – must be indented

- Python assigns car to be the first item in the list and then executes the indented statement(s)
- Then it gets the next item, assigns it to car, and executes the indented statement(s) again
- It repeats until all the items in list have been processed

For loops

Suppose we had this code:

```
cars = ['Kia', 'Honda', 'Toyota', 'Ford']
for car in cars:
    print(car + ' ' + str(len(car)))
```

The for loop would output this:

```
Kia 3
Honda 5
Toyota 6
Ford 4
```

Note that **len(car)** gives the length of each car string in the list as that car is "visited"

- **len(cars)** would give what?

15

Example: sum()

Consider a function that computes the sum of the numbers in a list

- Note this function exists in Python, named sum(), but by thinking how to write it we can better understand for loops.

First, initialize a variable **total** to zero

Then, use a **for loop** to add each number in the list to **total**

After all items have been added, the loop will terminate, and the function returns the final value of **total**

16

Example: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total
```

Initialize a variable to store the running total

```
# Example

t = sum([3, 5, 1])      # t will equal 9
```

See sum_tests.py

17

Example: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total
```

Visit each number in the list of numbers

```
# Example

t = sum([3, 5, 1])      # t will equal 9
```


18

Example: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total
```

Add each number to the running total

```
# Example

t = sum([3, 5, 1])      # t will equal 9
```


Example: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total
```

Return the final total

```
# Example

t = sum([3, 5, 1])      # t will equal 9
```

Example: sum()

Now we will ***trace*** the execution of this code to understand it better

A blue arrow will indicate the current line of code being executed

A table of values will show how the variables change value over time

21

Trace execution: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total
```

Variable	Value
total	0

Example

```
t = sum([3, 5, 1])      # t will equal 9
```


22

Trace execution: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total
```

Variable	Value
total	0
num	3

Example

t = sum([3, 5, 1]) # t will equal 9

Trace execution: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total
```

Variable	Value
total	3
num	3

Example

t = sum([3, 5, 1]) # t will equal 9

Trace execution: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total
```

Variable	Value
total	3
num	5

Example

t = sum([3, 5, 1]) # t will equal 9

25

Trace execution: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total
```

Variable	Value
total	8
num	5

Example

t = sum([3, 5, 1]) # t will equal 9

26

Trace execution: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total

# Example

t = sum([3, 5, 1])      # t will equal 9
```

Variable	Value
total	8
num	1

Trace execution: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total

# Example

t = sum([3, 5, 1])      # t will equal 9
```

Variable	Value
total	9
num	1

Trace execution: sum()

```
def sum(nums):
    total = 0
    for num in nums:
        total += num
    return total
```

Variable	Value
total	9
num	1

Example

```
t = sum([3, 5, 1])      # t will equal 9
```


Trace execution in PyCharm

PyCharm features a powerful tool called a **debugger** which can help trace the execution of a program

- Usually a debugger is used to help find bugs

First, set a **breakpoint** by clicking the mouse to the left of the line where the computer should pause execution

In sum_tests.py, put a breakpoint on line 8

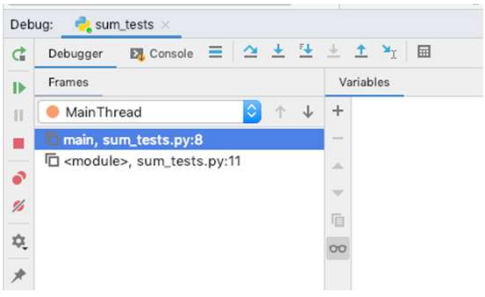
Trace execution in PyCharm

```
sum_tests.py
1  def sum(nums):
2      total = 0
3      for num in nums:
4          total += num
5      return total
6
7  def main():
8      scores = [3, 5, 1]
9      print('Sum of scores[]: ' + str(sum(scores)))
10
11 main()
```

- When the computer is commanded to debug the program, it will stop at that line with the breakpoint and not execute that line until it is told to
- When running the debugger for the first time, PyCharm may indicate some updates should be installed. If so, install the recommended updates

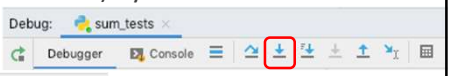
Trace execution in PyCharm

- To begin execution, right-click on sum_tests.py and pick "Debug 'sum_tests'". The computer stops at line 8.
- A "Debugger" panel opens
  - On the right, there is a sub-panel named "Variables" that will show the values of variables as the program runs



Trace execution in PyCharm

- Every time the downward facing blue arrow is hit (Step over button) PyCharm executes another line of code
- PyCharm highlights in blue what line it will execute *next*



sum_tests.py

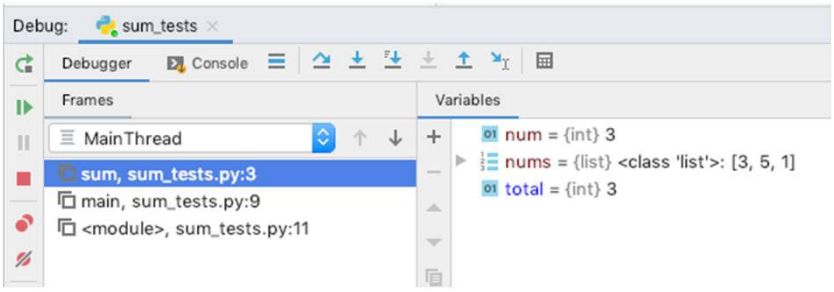
```
1  def sum(nums):  nums: <class 'list'>: [3, 5, 1]
2      total = 0   total: 3
3      for num in nums:  num: 3
4          total += num
5      return total
6
7  def main():
8      scores = [3, 5, 1]
9      print('Sum of scores[]: ' + str(sum(scores)))
```

The debugger shows the value of each variable in the source code

33

Trace execution in PyCharm

Here's the state of the program after hitting the blue arrow several times:



In lab, there will be opportunity to practice using the debugger
- [Hint] Getting familiar with this tool will save hours of headaches later on

34

List indexes

- Often an item in the middle of a list is needed
- If a list has n item, the locations in the list are numbered from 0 to n-1
  (not 1 through n)
- The notation **a[i]** stands for "the item at location i in list a"

- In programming, use the word **index** to refer to the numerical position of an element in a list
- Example: **scores = [89, 78, 92, 63, 92]**
  scores[0] is 89
  scores[2] is 92
  scores[5] gives an "index out of range" error (why?)

35

List indexes

The **index** method will indicate the position of an element in a list

If the requested element is not in the list, the Python interpreter will generate an error

Example:

scores = [89, 78, 92, 63, 92]

- **scores.index(92)** is 2, the index of the first occurrence of 92 in the scores list
- **scores.index(99)** generates this error: "ValueError: 99 is not in list"

36

List indexes

- If the program needs the index of a value, and it is not guaranteed the value is in the list, use an if statement in conjunction with the **in** operator to first make sure the item is actually in the list
- Example:

```
vowels = ['a', 'e', 'i', 'o', 'u']
letter = 'e'
if letter in vowels:
   print('That letter is at index ' + str(vowels.index(letter)) + '.')
else:
   print('That letter is not in the list.')
```

- Output: **That letter is at index 1.**

Iteration using list indexes

- A common programming "idiom" uses a for loop based on a list index:

```
for i in range(n):
   # do something with i
```

- **range(n)** means "the sequence of integers starting from zero and ranging up to, but not including, **n**"
- Python executes the body of the loop **n** times
- **i** is set to every value between 0 and **n-1** (**n** is NOT included)

Iteration using list indexes

This function computes and returns the sum of the first **k** values in a list (see partial_total.py)

```
def partial_total(nums, k):
  total = 0
  for i in range(k):
    total += nums[i]
  return total
```

Initialize the variable to store the running total

Example:

```
a = [4, 2, 8, 3, 1]
partial_total(a, 3)  # returns the value 14
partial_total(a, 1)  # returns the value 4
partial_total(a, 6)  # error
```


Iteration using list indexes

This function computes and returns the sum of the first **k** values in a list

```
def partial_total(nums, k):
  total = 0
  for i in range(k):
    total += nums[i]
  return total
```

Generate indexes 0 through *k*-1

Example:

```
a = [4, 2, 8, 3, 1]
partial_total(a, 3)  # returns the value 14
partial_total(a, 1)  # returns the value 4
partial_total(a, 6)  # error
```

Iteration using list indexes

This function computes and returns the sum of the first **k** values in a list

```
def partial_total(nums, k):
  total = 0
  for i in range(k):
    total += nums[i]
  return total
```

Add each number to the running total

Example:

```
a = [4, 2, 8, 3, 1]
partial_total(a, 3)  # returns the value 14
partial_total(a, 1)  # returns the value 4
partial_total(a, 6)  # error
```


41

Iteration using list indexes

This function computes and returns the sum of the first **k** values in a list

```
def partial_total(nums, k):
  total = 0
  for i in range(k):
    total += nums[i]
  return total
```

Return the final total

Example:

```
a = [4, 2, 8, 3, 1]
partial_total(a, 3)  # returns the value 14
partial_total(a, 1)  # returns the value 4
partial_total(a, 6)  # error
```


42

Iteration using list indexes

- Trace the execution of this function for one example

```
def partial_total(nums, k):
    total = 0
    for i in range(k):
        total += nums[i]
    return total
```

Variable	Value
total	0

- Example:

a = [4, 2, 8, 3, 1]
partial_total(a, 3) # returns the value 14

Iteration using list indexes

- Trace the execution of this function for one example

```
def partial_total(nums, k):
    total = 0
    for i in range(k):
        total += nums[i]
    return total
```

Variable	Value
total	0
i	0

- Example:

a = [4, 2, 8, 3, 1]
partial_total(a, 3) # returns the value 14

Iteration using list indexes

- Trace the execution of this function for one example

```
def partial_total(nums, k):
    total = 0
    for i in range(k):
        total += nums[i]
    return total
```

Variable	Value
total	4
i	0
nums[i]	4

- Example:

a = [4, 2, 8, 3, 1]
partial_total(a, 3) # returns the value 14

Iteration using list indexes

- Trace the execution of this function for one example

```
def partial_total(nums, k):
    total = 0
    for i in range(k):
        total += nums[i]
    return total
```

Variable	Value
total	4
i	1
nums[i]	4

- Example:

a = [4, 2, 8, 3, 1]
partial_total(a, 3) # returns the value 14

Iteration using list indexes

- Trace the execution of this function for one example

```
def partial_total(nums, k):
    total = 0
    for i in range(k):
        total += nums[i]
    return total
```

Variable	Value
total	6
i	1
nums[i]	2

- Example:

a = [4, 2, 8, 3, 1]
partial_total(a, 3) # returns the value 14

47

Iteration using list indexes

- Trace the execution of this function for one example

```
def partial_total(nums, k):
    total = 0
    for i in range(k):
        total += nums[i]
    return total
```

Variable	Value
total	6
i	2
nums[i]	2

- Example:

a = [4, 2, 8, 3, 1]
partial_total(a, 3) # returns the value 14

48

Iteration using list indexes

- Trace the execution of this function for one example

```
def partial_total(nums, k):
    total = 0
    for i in range(k):
        total += nums[i]
    return total
```

Variable	Value
total	14
i	2
nums[i]	8

- Example:

a = [4, 2, 8, 3, 1]
partial_total(a, 3) # returns the value 14

Iteration using list indexes

- Trace the execution of this function for one example

```
def partial_total(nums, k):
    total = 0
    for i in range(k):
        total += nums[i]
    return total
```

Variable	Value
total	14
i	2
nums[i]	8

- Example:

a = [4, 2, 8, 3, 1]
partial_total(a, 3) # returns the value 14

String indexes

Strings and lists have much in common, including indexing:

name[i] would give us the character at index **i** of the string **name**

nums[i] gives us the element at index **i** of the list **nums**

Examples:

```
title = 'Lord of the Rings'
print(title[0])      # prints L
print(title[2])      # prints r
j = 6
print(title[j])      # prints f
```


Making lists of numbers

- The **range** function can be used to make a list of integers
- This example makes a list of the numbers from 0 to 9:

```
nums = list(range(10))
```

- Note that **list** is the name of a **class** in Python
  - A class describes what kinds of data an object can store
- In general, if a class name is used as a function, Python will create an object of that class
  - For example, **list()** or **str(50)**
  - These functions are called **constructors** because they construct new objects
  - More on this topic later in the course

Back to the Sieve algorithm

We now know how to make a list of prime numbers

Use a Python **list** object to represent a "worksheet" of numbers that will be progressively crossed off

The list will initially have all the integers from 2 to n (the upper limit)

Will use for loops to iterate over the list to cross off composite numbers

- Can pass two values to range – e.g. **range(2, 100)**
  - The first value is the lower limit (2 in the example)
  - The other as the upper limit, minus 1 (99 in the example)
  - So to make a list of numbers between 2 and 99, type **list(range(2, 100))**

Back to the Sieve algorithm

- The steps of the algorithm are easier to understand if two "placeholder" values are added at the front of the list to represent 0 and 1 (neither of which is a prime number)

- Python has a special value called **None** that stands for "no object"
- Since the expression **a + b** means "concatenate **a** and **b**" where **a** and **b** are lists, the statement below creates the initial worksheet:

```
worksheet = [None, None] + list(range(2,100))
```

- With the two placeholders at the front, any number **i** will be at **worksheet[i]**
  - For example, the number 5 will be at **worksheet[5]** instead of **worksheet[3]**

PythonLabs

PythonLabs is a set of Python modules developed for the course textbook

PythonLabs homepage: http://ix.cs.uoregon.edu/~conery/eic/python/

Installation instructions: http://ix.cs.uoregon.edu/~conery/eic/python/installation.html

55

PythonLabs – SieveLab

The module for the Sieve algorithm is named SieveLab

SieveLab has:

- A complete implementation of a **sieve** function for finding prime numbers
- Functions that use algorithm animation to generate graphical displays to show how the algorithm works

56

SieveLab

Below you can see an example of how to use the SieveLab module

```
import PythonLabs.SieveLab
worksheet = [None, None] + list(range(2, 400))
PythonLabs.SieveLab.view_sieve(worksheet)
```

Call a SieveLab function named **mark_multiples** to see how the algorithm removes multiples of a specified value

- The two arguments to mark_multiples are a number k and the worksheet list
- The screen will be updated to show that k is prime (indicated by a blue square)
- Gray boxes will be drawn over all the multiples of k

57

SieveLab

PythonLabs.SieveLab.mark_multiples(2, worksheet)

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139
140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179
180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199
200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219
220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259
260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279
280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299
300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319
320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339
340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359
360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379
380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399

58

SieveLab

Call SieveLab's **erase_multiples** function to erase the marked numbers

- Erase the multiples of 2 using this function

SieveLab

PythonLabs.SieveLab.erase_multiples(2, worksheet)

	2	3	5	7	9	11	13	15	17	19
21	23	25	27	29	31	33	35	37	39	
41	43	45	47	49	51	53	55	57	59	
61	63	65	67	69	71	73	75	77	79	
81	83	85	87	89	91	93	95	97	99	
101	103	105	107	109	111	113	115	117	119	
121	123	125	127	129	131	133	135	137	139	
141	143	145	147	149	151	153	155	157	159	
161	163	165	167	169	171	173	175	177	179	
181	183	185	187	189	191	193	195	197	199	
201	203	205	207	209	211	213	215	217	219	
221	223	225	227	229	231	233	235	237	239	
241	243	245	247	249	251	253	255	257	259	
261	263	265	267	269	271	273	275	277	279	
281	283	285	287	289	291	293	295	297	299	
301	303	305	307	309	311	313	315	317	319	
321	323	325	327	329	331	333	335	337	339	
341	343	345	347	349	351	353	355	357	359	
361	363	365	367	369	371	373	375	377	379	
381	383	385	387	389	391	393	395	397	399	

SieveLab

After erasing multiples of 2, the lowest unmarked number is 3, so on the next round, remove multiples of 3

Repeat the “marking” and “erasing” steps until only prime numbers are left

Following is the process for marking and erasing multiples of 3, 5 and 7

SieveLab

PythonLabs.SieveLab.mark_multiples(3, worksheet)

	2	3	5	7	9	11	13	15	17	19
21	23	25	27	29	31	33	35	37	39	
41	43	45	47	49	51	53	55	57	59	
61	63	65	67	69	71	73	75	77	79	
81	83	85	87	89	91	93	95	97	99	
101	103	105	107	109	111	113	115	117	119	
121	123	125	127	129	131	133	135	137	139	
141	143	145	147	149	151	153	155	157	159	
161	163	165	167	169	171	173	175	177	179	
181	183	185	187	189	191	193	195	197	199	
201	203	205	207	209	211	213	215	217	219	
221	223	225	227	229	231	233	235	237	239	
241	243	245	247	249	251	253	255	257	259	
261	263	265	267	269	271	273	275	277	279	
281	283	285	287	289	291	293	295	297	299	
301	303	305	307	309	311	313	315	317	319	
321	323	325	327	329	331	333	335	337	339	
341	343	345	347	349	351	353	355	357	359	
361	363	365	367	369	371	373	375	377	379	
381	383	385	387	389	391	393	395	397	399	

SieveLab

PythonLabs.SieveLab.erase_multiples(3, worksheet)

	2	3	5	7		11	13		17	19
		23	25		29	31		35	37	
41		43		47	49		53	55		59
61			65	67		71	73		77	79
		83	85		89	91		95	97	
101		103		107	109		113	115		119
121			125	127		131	133		137	139
		143	145		149	151		155	157	
161		163		167	169		173	175		179
181			185	187		191	193		197	199
		203	205		209	211		215	217	
221		223		227	229		233	235		239
241			245	247		251	253		257	259
		263	265		269	271		275	277	
281		283		287	289		293	295		299
301			305	307		311	313		317	319
		323	325		329	331		335	337	
341		343		347	349		353	355		359
361			365	367		371	373		377	379
		383	385		389	391		395	397	

63

SieveLab

PythonLabs.SieveLab.mark_multiples(5, worksheet)

	2	3	5	7		11	13		17	19
		23	25		29	31		35	37	
41		43		47	49		53	55		59
61			65	67		71	73		77	79
		83	85		89	91		95	97	
101		103		107	109		113	115		119
121			125	127		131	133		137	139
		143	145		149	151		155	157	
161		163		167	169		173	175		179
181			185	187		191	193		197	199
		203	205		209	211		215	217	
221		223		227	229		233	235		239
241			245	247		251	253		257	259
		263	265		269	271		275	277	
281		283		287	289		293	295		299
301			305	307		311	313		317	319
		323	325		329	331		335	337	
341		343		347	349		353	355		359
361			365	367		371	373		377	379
		383	385		389	391		395	397	

64

SieveLab

PythonLabs.SieveLab.erase_multiples(5, worksheet)

2	3	5	7		11	13		17	19
	23			29	31			37	
41	43		47	49		53			59
61			67		71	73		77	79
	83			89	91			97	
101	103		107	109		113			119
121			127		131	133		137	139
	143			149	151			157	
161	163		167	169		173			179
181			187		191	193		197	199
	203			209	211			217	
221	223		227	229		233			239
241			247		251	253		257	259
	263			269	271			277	
281	283		287	289		293			299
301			307		311	313		317	319
	323			329	331			337	
341	343		347	349		353			359
361			367		371	373		377	379
	383			389	391			397	

65

SieveLab

PythonLabs.SieveLab.mark_multiples(7, worksheet)

2	3	5	7		11	13		17	19
	23			29	31			37	
41	43		47	49		53			59
61			67		71	73		77	79
	83			89	91			97	
101	103		107	109		113			119
121			127		131	133		137	139
	143			149	151			157	
161	163		167	169		173			179
181			187		191	193		197	199
	203			209	211			217	
221	223		227	229		233			239
241			247		251	253		257	259
	263			269	271			277	
281	283		287	289		293			299
301			307		311	313		317	319
	323			329	331			337	
341	343		347	349		353			359
361			367		371	373		377	379
	383			389	391			397	

66

SieveLab

PythonLabs.SieveLab.erase_multiples(7, worksheet)

	[2]	[3]	[5]	[7]		11	13	17	19
		23			29	31		37	
41		43		47			53		59
61				67		71	73		79
		83			89			97	
101		103		107	109		113		
121				127		131		137	139
		143			149	151		157	
		163		167	169		173		179
181				187		191	193	197	199
					209	211			
221		223		227	229		233		239
241				247		251	253	257	
		263			269	271		277	
281		283			289		293		299
				307		311	313	317	319
		323				331		337	
341				347	349		353		359
361				367			373	377	379
		383			389	391		397	

Sieve algorithm: a helper function

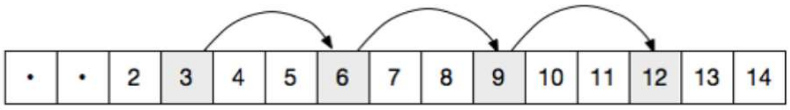
- An important step toward implementing the Sieve algorithm is to write a function that solves a small part of the problem

- The function **sift** will make a single pass through the worksheet
- Pass it a number **k**, and **sift** will find and remove multiples of **k**
- For example, to sift out multiples of 5 from the list called **worksheet** we could write:
  sift(5, worksheet)

- **sift** has a very specific purpose, and it is unlikely to be used except as part of an implementation of the Sieve algorithm
  - Programmers call special-purpose functions like this **helper functions**

Stepping through the worksheet

- Each call to sift is used to find multiples of k
- The first one is 2*k
- Notice that the remaining multiples (3*k, 4*k, etc) are all k steps apart:



- Use a for-loop with a **range** expression to walk through the list:

  for i in range (2*k, len(a), k):
- Note this **range** expression has three arguments:
  1. the starting point
  2. the ending point
  3. the **step size (k)**

Stepping through the worksheet

To remove a number from the worksheet, we could use the Python **del** statement, which deletes an item from a list

- But this would shorten the list and make it harder to walk through on future iterations

A better solution: replace the items with placeholders (**None** objects)

The complete implementation of the sift function:

```
def sift(k, a):
    for i in range(2*k, len(a), k):
        a[i] = None
```

Stepping through the worksheet

```
def sift(k, a):
    for i in range(2*k, len(a), k):
        a[i] = None
```

- An example of **sift** in action:

  worksheet = [None, None] + list(range(2, 16))
- **worksheet** is now:

  [None, None, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
- Now call **sift(2, worksheet)**
- **worksheet** becomes this:

  [None, None, 2, 3, None, 5, None, 7, None, 9, None, 11, None, 13, None, 15]

The sieve() function

- Now that there is a helper function to do the hard work, we want to write the **sieve** to solve the complete problem
- Much easier to write now that we have the helper function written
- When a program has helpers, a function like **sieve** (which is called to solve the complete problem) is known as a **top-level function**

The sieve() function

- Goal: Write a loop that starts by sifting multiples of 2 and keep calling **sift** until all composite numbers are removed

- This loop can stop when the next number to send to **sift** is greater than the *square root of* ***n*** (why?)
- Thus, the for loop that controls the loop should set **k** to every value from 2 up to the square root of **n**:

```
for k in range(2, sqrt(n)):
```


The sieve() function

```
for k in range(2, sqrt(n)):
```

There is a problem with this code: Can not pass a floating-point value to **range**

We can "round up" the square root (e.g. 17.2 -> 18)
- That provides what is needed: an integer greater than the highest possible prime factor of n

A function named **ceil** in Python's math library does this operation
- ceil is short for "ceiling"

A corresponding function named **floor** rounds a floating-point value down to the nearest integer

sieve()'s main loop

- One important detail: before sifting out multiples of a number, make sure it hasn't already been removed
- For example, don't need to sift multiples of 4 because 4 was already removed when sifting multiples of 2
  - **sift** would still work, but the program would be less efficient
- The main loop looks like this:

```
for k in range(2, ceil(sqrt(n))):
    if worksheet[k] is not None:
        sift(k, worksheet)
```

- Note that the expression **x is not None** is the preferred way of testing to see if **x** is a reference to the **None** object

Sieve: remove the placeholders

- One last step: to make the final list, remove the **None** objects from the worksheet
- We can make a new helper function called **non_nulls** returns a copy of the worksheet, but without any **None** objects
- It makes an initial empty list named **res** (for "result")
- Then it uses a for loop to look at every item in the input list
- If an item is not **None**, the item is appended to **res** using the **append** method for lists
- When the iteration is complete, **res** is returned as the result of the function call

Sieve: remove the placeholders

```
def non_nulls(a):
    res = []
    for x in a:
        if x is not None:
            res.append(x)
    return res
```

Initialize `res[]` to be the empty list

- Example:

```
worksheet = [None, None, 2, 3, None, 5, None, 7, None, None,
             None, 11, None, 13, None, None]
worksheet = non_nulls(worksheet)    # worksheet is now: [2, 3, 5, 7, 11, 13]
```


Sieve: remove the placeholders

```
def non_nulls(a):
    res = []
    for x in a:
        if x is not None:
            res.append(x)
    return res
```

Visit each element in the list `a[]`

- Example:

```
worksheet = [None, None, 2, 3, None, 5, None, 7, None, None,
             None, 11, None, 13, None, None]
worksheet = non_nulls(worksheet)    # worksheet is now: [2, 3, 5, 7, 11, 13]
```

Sieve: remove the placeholders

```
def non_nulls(a):
    res = []
    for x in a:
        if x is not None:
            res.append(x)
    return res
```

See if **x** is actually a number

- Example:

```
worksheet = [None, None, 2, 3, None, 5, None, 7, None, None,
             None, 11, None, 13, None, None]
worksheet = non_nulls(worksheet)    # worksheet is now: [2, 3, 5, 7, 11, 13]
```


Sieve: remove the placeholders

```
def non_nulls(a):
    res = []
    for x in a:
        if x is not None:
            res.append(x)
    return res
```

If **x** is a number, append it to `res[]`

- Example:

```
worksheet = [None, None, 2, 3, None, 5, None, 7, None, None,
             None, 11, None, 13, None, None]
worksheet = non_nulls(worksheet)    # worksheet is now: [2, 3, 5, 7, 11, 13]
```

Aside: appending to a List

+= can be used to concatenate one string to the end of another
- This syntax can also be used to append one list to another

Example:

```
fruits = ['apple', 'orange']

fruits += ['banana', 'mango', 'pear']
# fruits is now: ['apple', 'orange', 'banana', 'mango', 'pear']

fruits += ['pineapple']
# fruits is now: ['apple', 'orange', 'banana', 'mango', 'pear', 'pineapple']
```


The Sieve algorithm: completed!

- Now, put all the pieces together
- Import the **math** library to get access to **sqrt** and **ceil**
- In the body of the **sieve** function:
  - Create the **worksheet** with two initial **None** objects and all integers from 2 to **n**
  - Add the for-loop that calls **sift**
  - Call **non_nulls** to remove the **None** objects from the **worksheet**
- See sieve.py and the next slide for the code
- See PythonLabs/SieveLab.py: lines 12–28 for the textbook's implementation of the **sieve** function
- Run sieve_visualization.py using PyCharm's Python Console to see it in action

Completed sieve() function

```
from math import *
def sift(k, a):
   ... # see earlier slides

def non_nulls(a):
   ... # see earlier slides

def sieve(n):
   worksheet = [None, None] + list(range(2, n))
   for k in range(2, ceil(sqrt(n))):
      if worksheet[k] is not None:
         sift(k, worksheet)
   return non_nulls(worksheet)

primes = sieve(100)
print(primes)
```

See sieve.py

Abstraction

Now we have a function for making lists of prime numbers, which can be saved and used later

It can be used to answer questions about primes, such as:

- How many primes are less than n?
- What is the largest gap between successive primes?
- What are some twin primes (two prime numbers that differ only by 2, like 17 and 19)?

This is a good example of **abstraction**: There is a nice, neat package that can be saved and **reused**

In the future, there is no need to worry about the implementation details of sieve: just use it!

- Just need to know that **sieve(n)** makes a list of prime numbers from 2 to n

Additional examples

Next is a look at some additional examples of how to use for loops and lists to solve problems in Python

85

Example: find the maximum

Try writing an algorithm to find the maximum value in a list

- Note that a function already exists in Python (called **max**), but it is good practice

The basic idea is to *iterate* over the list and keep track of the largest value seen to that point

Begin by taking the value at index 0 as the maximum

Continue with the remainder of the list, comparing the next value with the current maximum and updating the maximum if and when a larger value than the current maximum is found

86

Example: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```


Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	20

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	20
i	1

89

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	20
i	1
nums[i]	16

90

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:  # False
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	20
i	1
nums[i]	16

91

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	20
i	2
nums[i]	22

92

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	20
i	2
nums[i]	22

93

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:    # True
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	20
i	2
nums[i]	22

94

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	22
i	2
nums[i]	22

95

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	22
i	3
nums[i]	30

96

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	22
i	3
nums[i]	30

97

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:      # True
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	22
i	3
nums[i]	30

98

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	30
i	3
nums[i]	30

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	30
i	4
nums[i]	17

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	30
i	4
nums[i]	17

101

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:    # False
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	30
i	4
nums[i]	17

102

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	30
i	5
nums[i]	24

103

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	30
i	5
nums[i]	24

104

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:     # False
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	30
i	5
nums[i]	24

105

Trace execution: find_max.py

```
def find_max(nums):
    maximum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > maximum:
            maximum = nums[i]
    return maximum

ages = [20, 16, 22, 30, 17, 24]
max_age = find_max(ages) # max_age will be 30
print('Maximum age: ' + str(max_age))
```

Variable	Value
maximum	30
i	5
nums[i]	24

106

Example: count the vowels

A for loop can be used to iterate over the characters of a string

To see how this works, consider a function called **count_vowels** that counts the number of vowels (lowercase or uppercase) in a word

- To make this problem a little easier to solve, we can call the **lower()** method for strings, which makes a copy of a given string and changes all the uppercase letters to lowercase
  - upper() makes all letters uppercase

Strings are **immutable** (unchangeable) objects

To convert a string into lowercase we must make a lowercase copy of it and replace the original string with the new one

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower(): # search through a
        if letter in vowels:    # lowercase copy of
            num_vowels += 1     # the original word
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	0

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	0
letter	c

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	0
letter	c

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:        # False
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	0
letter	c

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	0
letter	i

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	0
letter	i

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:        # True
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	0
letter	i

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	1
letter	i

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	1
letter	d

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	1
letter	d

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:        # False
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	1
letter	d

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	1
letter	e

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	1
letter	e

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:          # True
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	1
letter	e

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	2
letter	e

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	2
letter	r

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	2
letter	r

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:          # False
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	2
letter	r

Example: vowels.py

```
def count_vowels(word):
    vowels = 'aeiou'
    num_vowels = 0
    for letter in word.lower():
        if letter in vowels:
            num_vowels += 1
    return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Variable	Value
num_vowels	2
letter	r

A list of lists

In Python, a list can contain objects of any type

A list is an object. Therefore, a list can contain other lists!

Imagine there is a group of 4 students, and for each student there are 3 exam scores:

```
scores = [[89, 85, 90], [78, 85, 72],
          [99, 86, 92], [82, 84, 79]]
```

To access a particular score, two indices are needed:

- First, which students grade is needed (0 through 3)
- Second, which score of that student is desired (0 through 2)

Example: scores[3][1] is fourth student's score on the second exam (which is 84)

Example: compute averages (v1)

We want to write code that will compute the average score that students earned on each exam

Will write more than one version of the program ➔ But start simple

In the first version we will "hard-code" several values (the number of students and the number of scores) in the program

Then, generalize things a bit and use variables for these values

Example: averages_v1.py

```
# scores represents 4 students who each took 3 exams
scores = [[89, 85, 90], [78, 85, 72],
          [99, 86, 92], [82, 84, 79]]

averages = [0, 0, 0]

for student in scores:
   averages[0] += student[0]
   averages[1] += student[1]
   averages[2] += student[2]

for i in range(3):
   averages[i] /= 4

print(averages)
```


131

Example: compute averages (v2)

The first version of the code has a major negative: the algorithm will work only for a class of four students who took three exams

Suppose the class is larger or smaller? Or suppose the students took more or fewer exams?

132

Example: compute averages (v2)

Next development attempt is a better (but more complicated) version of the algorithm that can adapt to larger/smaller class sizes and more/fewer exams

The approach will rely on **nested loops**, which means there will be one loop inside of another

Nested loops will become increasingly important as the course progresses

Example: averages_v2.py

One other thing before looking at the program

Recall that syntax like **'Hi'*3** will create a new string by repeating a given string a desired number of times

- For instance, **'Hi'*3** equals **'HiHiHi'**
- In a similar manner, **[0]*3** would create a list containing 3 zeroes, namely, **[0, 0, 0]**

Thus, the * notation with strings and lists is essentially a form of **concatenation**

Example: averages_v2.py

```
scores = [[89, 85, 90], [78, 85, 72], [99, 86, 92], [82, 84, 79]]

num_students = len(scores)
num_exams = len(scores[0])          # each student took the
averages = [0] * num_exams          # same number of exams

for student in scores:
    for i in range(0, num_exams):   # nested loops
        averages[i] += student[i]

for i in range(0, num_exams):
    averages[i] /= num_students

print(averages)
```


135

Example: compute averages (v3)

In a third and final version of the exam average calculator, the computations will be *encapsulated* (enclosed or wrapped) inside of a function

compute_averages(students)

The function takes the list of scores as its argument

After computing the exam averages, the function returns a list of the average scores

This illustrates that Python functions can return many values at once (via a list), not just a single number or string

136

Example: averages_v3.py

```
scores = [[89, 85, 90], [78, 85, 72], [99, 86, 92], [82, 84, 79]]

def compute_averages(students):
    num_students = len(students)
    num_exams = len(students[0])
    avgs = [0] * num_exams

    for student in students:
        for i in range(0, num_exams):
            avgs[i] += student[i]

    for i in range(0, num_exams):
        avgs[i] /= num_students

    return avgs

averages = compute_averages(scores)
print(averages)
```


Example: bottles of beer/milk

The final example is on a lighter note looking at a program that prints the lyrics of the song “99 Bottles of Beer on the Wall”

- In this song, the singer needs to count from 99 down to 0

The **range** command can be used to count up, but it also can count down if given a negative number for the step size

For example, **range(10,-1,-1)** will count down from 10 to 0 by 1s

So **list(range(10,-1,-1))** would generate the list [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

The code on the next slide asks the user for the starting number so that the program can start from a value other than 99

Example: bottles.py

```
age = int(input('How old are you? '))

if age < 21:
   drink_type = 'milk'
else:
   drink_type = 'beer';

num_bottles = int(input('How many bottles of ' + drink_type + ' do you have? '))

for bottle in range(num_bottles, -1, -1):
   if bottle > 1:
      print(str(bottle) + ' bottles of ' + drink_type +
          ' on the wall!')
   elif bottle == 1:
      print('1 bottle of ' + drink_type + ' on the wall!')
   else:
      print('No bottles of ' + drink_type + ' on the wall!')
```


Example: vowels.py

```
def count_vowels(word):
   vowels = 'aeiou'
   num_vowels = 0
   for letter in word.lower(): # search through a
      if letter in vowels:     # lowercase copy of
         num_vowels += 1       # the original word
   return num_vowels

word = 'Cider'
print('The number of vowels in ' + word + ' is ' +
str(count_vowels(word))) # will print 2
```

Modify this program to:
1. Also count and return the number of non-vowel letters
– Hint: use a list to return both numbers

2. Print out the number of vowels and non-vowels
- Hint: need to access the index of the returned list

Example: bottles.py

```
age = int(input('How old are you? '))

if age < 21:
   drink_type = 'milk'
else:
   drink_type = 'beer';

num_bottles = int(input('How many bottles of '
+ drink_type + ' do you have? '))

for bottle in range(num_bottles, -1, -1):
   if bottle > 1:
      print(str(bottle) + ' bottles of ' + drink_type +
          ' on the wall!')
   elif bottle == 1:
      print('1 bottle of ' + drink_type + ' on the wall!')
   else:
      print('No bottles of ' + drink_type + ' on the wall!')
```

Modify this program to:
1. Write milk only for people younger than the legal drinking age in your country

2 Make it count up from 1 to the user input number, incrementing by 2 (e.g. 1, 3, 5…)

Questions?