Road Map

Algorithm Design and Analysis

- Week 1: Intro, Counting, Programming
- This week → Week 2: Programming
- Week 3: Complexity
- Week 4: Iteration & Decomposition
- Week 5: Recursion

Fundamental Data Structures

(Weeks 6 - 10)

Computational Intractability and Heuristic Reasoning

(Weeks 11 - 13)

School of
Information Systems

SMU
SINGAPORE MANAGEMENT UNIVERSITY

COR-IS1702: COMPUTATIONAL THINKING
WEEK 2B: ALGORITHMS

This is going to be real fast!

You can always "fall back" on Chapter 3 of JSC :-)

2020/21 Term 1

Learning Outcomes

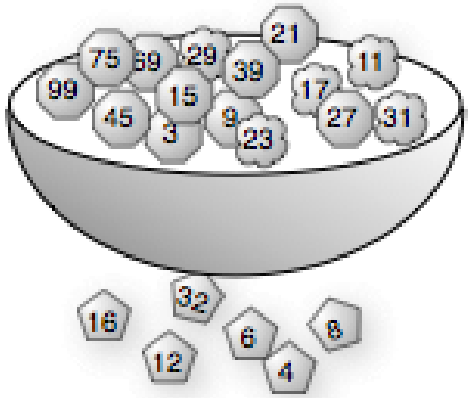
- By the end of this session, you should:

  - Be introduced to a simple algorithm for finding prime numbers
  - Be familiar with Python lists and how to write for loops
  - Be able to run a Python program from the terminal window

Problem: Prime Numbers

- A number is
  - ***prime*** if it cannot be written as the product of two smaller numbers. E.g.: 5, 11, 73, 9967
  - ***Composite*** if it is not a prime number. E.g. 10 (2 × 5), 99 (3 × 3 × 11)
  - 1 is not a prime number by definition; 2 is the smallest prime number.
- The Sieve of Eratosthenes is one of the oldest known algorithms
  - dates back to at least 200 BC
  - Used to return a list of prime numbers from 1 to n.

The Sieve Algorithm: Basic Idea

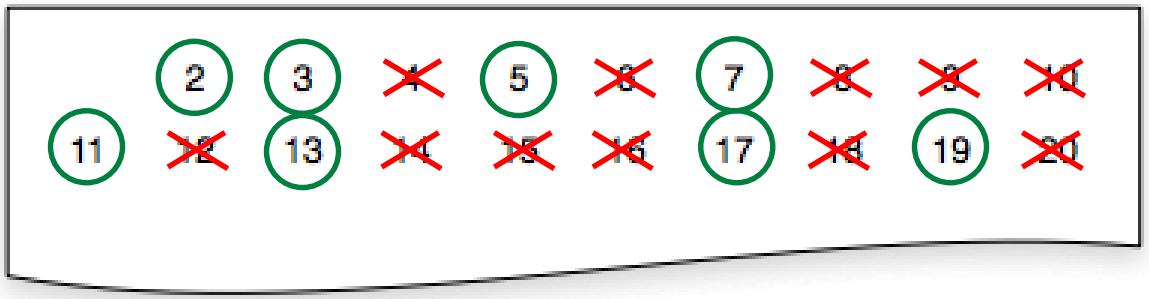


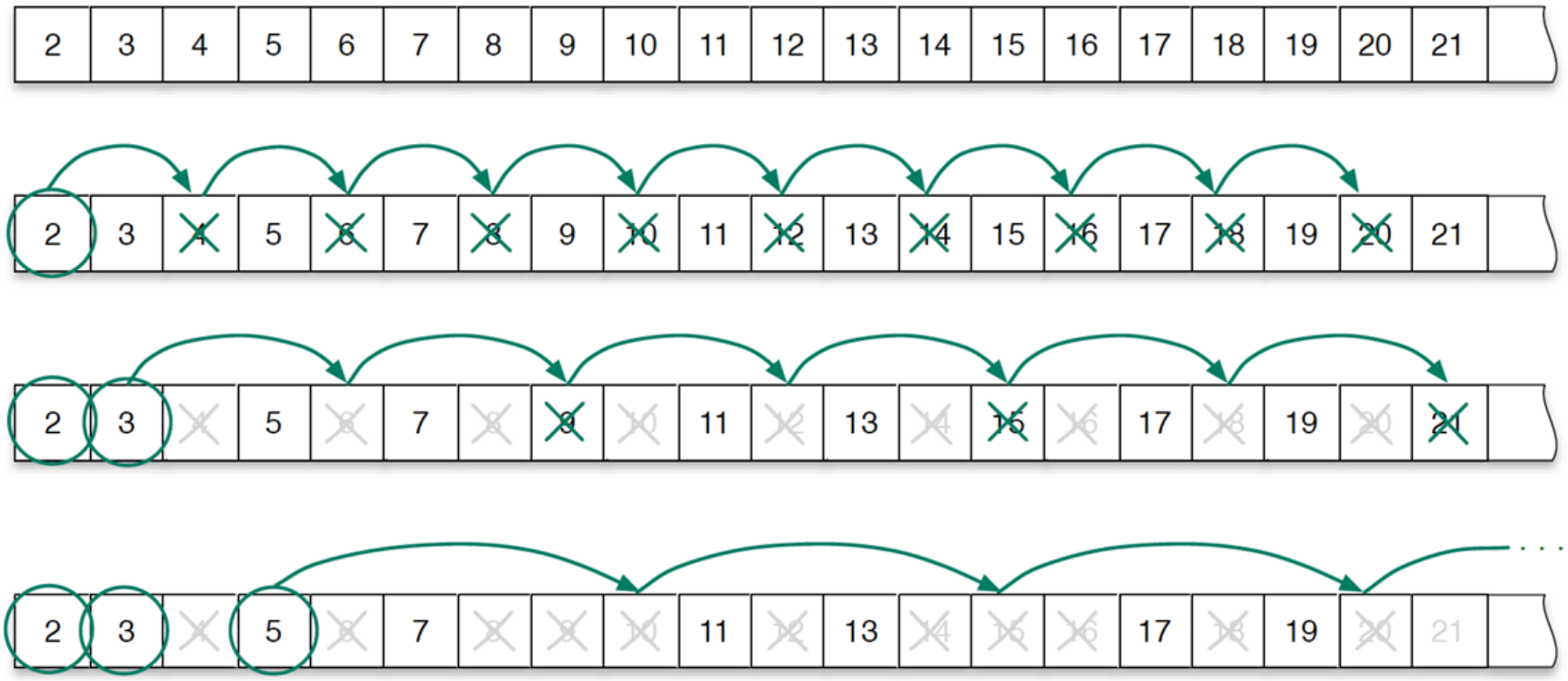
(1) Make a list of numbers from 2 to n

(2) Repeat until list is empty:

(2a) the 1st number in the list is prime

(2b) cross off multiples of the most recent prime





To find prime numbers write the integers starting from 2 on a strip of paper, then systematically cross off multiples of 2, multiples of 3, etc. *until only primes are left.*

Implementing an Algorithm as a Program

- The method described on the previous slide works well for short lists
- But what if you want to find prime numbers between 2 and 100? 1000?
  - it's a tedious process to write out a list of 100 numbers
  - takes a lot of paper to make a list of 1000 numbers
  - chances are you will make a few mistakes (boring job)
- Can we turn this algorithm into a computation?

At the end of this lesson, we will come up with a Python program that implements the sieve algorithm

We will come back to this problem later.

Lists in Python

- We have seen strings, integers, floats and Booleans. A list is also a data type, but it's special:

  A list stores multiple values (or elements). Each of these values can be a string, integer… etc.

```
>>> a = [1, 2, 3, 4, 5]
>>> type(a)
<class 'list'>
>>> len(a)
5
>>> b = ["apple", "orange", 3, None, False, "test"]
>>> type(b)
<class 'list'>
>>> len(b)
6
>>> c = []
>>> len(c)
0
>>> c.append(99)
>>> c
[99]
>>> c.append(88)
>>> c
[99, 88]
```

*When you pass a **string** to **len()**, it returns the no. of characters in the string. When you pass a **list** to **len()**, it returns the no. of elements in the **list**.*

*A **list** with different types of elements*
***None** is a special value meaning nothing.*

Inserting new elements at the back

In-class Ex: Chapter 3, T1-15

- Try Tutorial Project T1-2 on **p.72** of the PDF (p.58 of textbook)

  Import and use the **sieve()** function from the **PythonLabs.SieveLab** module, just to see what it returns.

- Try Tutorial Project T3-15 on **p.75** of the PDF (p.61 of textbook)

  Playing about with a list in Python

Is it possible for a list to contain another list?

A 2-dimension list? Try it!

How can you insert an element to the front of a list? To a specific position somewhere in the middle?

Iterating Through a List using **for**

- A Python keyword for looping: **for**
- Looping = repeat certain statements until a condition is met

Examples of for loops.

```
1  def print_list(a):
2      for x in a:
3          print(x)
```

A function that prints every item in a list.

```
1  def total(a):
2      sum = 0
3      for x in a:
4          sum += x
5      return sum
```

A function that computes the sum of a list of numbers Note that when there are two or more statements in the body of a loop they must all be indented the same number of spaces.

*Means: loop as many times as the number of elements in **a**. During each iteration (or loop), **x** will store the value of the current element.*

Same as:
sum = sum + x

In-class Ex: T16-26

- Try Tutorial Project T16-26 on **p.77** of the PDF (p.63 of textbook)

  Iterate through a list

  Write a **total()** function that does the same thing as the built-in **sum()** function. You will check if **total()** is working by comparing its return value with **sum()**.

- What happens if you pass to **sum()** (or **total()**) an array that contains non-integers (such as strings?) - try it!

  Is it possible to modify **total()** so that it will only consider elements in the passed-in list that are integers?

Accessing a Particular Element from a List

- Lists are "ordered", meaning each element has a specific position. The index of an element in the list is its position.
- Index numbering starts from 0 instead of 1.

```
>>> names = ["sonic", "tails", "amy", "knuckles"]
>>> names[0]
'sonic'
>>> names[2]
'amy'
>>> names[3]
'knuckles'
>>> names[4]
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    names[4]
IndexError: list index out of range
>>> names[-1]
'knuckles'
>>> names[-4]
'sonic'
>>> names[-5]
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    names[-5]
IndexError: list index out of range
>>> names[0] = "shadow"
>>> names
['shadow', 'tails', 'amy', 'knuckles']
```

Error

Assigning a new value to position 0

Using range(0, n)

- Another way to use for:

Means: the set of integers from 0 through (n-1)
e.g.: range(0, 5) means 0, 1, 2, 3, 4
range(3, 6) means 3, 4, 5

```
1  def partial_total(n,a):
2      """
3      Compute the total of the first
4      n items in list a.
5      """
6      sum = 0
7      for i in range(0,n):
8          sum += a[i]
9      return sum
```

During the 1st iteration, i will store 0,
During the 2nd iteration, i will store 1… etc
During the last iteration, i will store (n-1)
i is a local variable - sometimes called the loop counter

Using range(0, n, step)

Keeps cursor on the same line after printing

```
>>> for i in range(2,20):
        print(str(i)+" ", end="")


2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Range can take in a 3rd number:

Step = 2
*Means: loop counter **i** will increase by 2 during each iteration.*

```
>>> for i in range(2,20,2):
        print(str(i)+" ", end="")


2 4 6 8 10 12 14 16 18
```

In-class Ex: T27-39

- Try Tutorial Project T27-39 on p.82 of the PDF (p.68 of textbook)

  Use index to retrieve an element in a list

  Trying **for** with **range(x, y)** and **range(x, y, step)**

  "re" in notes --> returns **True** or **False**

  notes.index("re") --> returns the first position at which "re" is found in notes

Using **list()** to Create a List with Consecutive Integers

- Instead of

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

you can create a list like this as well:

```
a = list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Another e.g.:

```
b = list(range(2, 9))
[2, 3, 4, 5, 6, 7, 8]
```

- Create a list (worksheet) containing 100 elements:

```
worksheet = [None, None] + list(range(2, 100))
[None, None, 2, 3, 4, 5… 98, 99]
```

Code for **sieve()**, **sift()** and **non_nulls()**

```
from math import sqrt, ceil

def sieve(n):
    "Return a list of all prime numbers less than 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)

def sift(k, a):
    "Remove multiples of k from list a"
    for i in range(2*k, len(a), k):
        a[i] = None

def non_nulls(a):
    "Return a copy of list a with None objects removed"
    res = []
    for x in a:
        if x is not None:
            res.append(x)
    return res
```

Helper Function: **sift()**

```
14  def sift(k, a):
15      "Remove multiples of k from list a"
16      for i in range(2*k, len(a), k):
17          a[i] = None
```

- Takes in **k** (an integer) and **a** (a list of integers)
- "Marks" the elements at positions 2*k, 3*k, 4*k etc… as **None**

```
>>> l = list(range(0,10))
>>> sift(2, l)
>>> l
[0, 1, 2, 3, None, 5, None, 7, None, 9]
>>> j = list(range(0,10))
>>> sift(3, j)
>>> j
[0, 1, 2, 3, 4, 5, None, 7, 8, None]
```

Helper Function: **non_nulls()**

```
19  def non_nulls(a):
20      "Return a copy of list a with None objects removed"
21      res = []
22      for x in a:
23          if x is not None:
24              res.append(x)
25      return res
```

- Takes in a list of integers (**a**)
- Returns a new list that is similar to **a** except with all the **None** elements removed

```
>>> l1 = [99, None, None, 88, 7, None, 6, 5, None]
>>> non_nulls(l1)
[99, 88, 7, 6, 5]
>>> l
[0, 1, 2, 3, None, 5, None, 7, None, 9]
>>> non_nulls(l)
[0, 1, 2, 3, 5, 7, 9]
```

sieve() Function

```
3  def sieve(n):
4      "Return a list of all prime numbers less than n"
5
6      worksheet = [None, None] + list(range(2,n))
7
8      for k in range(2, ceil(sqrt(n))):
9          if worksheet[k] is not None:
10             sift(k, worksheet)
11
12     return non_nulls(worksheet)
```

- Line 6: **worksheet** is a list that contains:

  [None, None, 2, 3, 4, 5… 98, 99]

  Position 0 and 1 have been marked as **None**

  (by definition, 0 and 1 are not primes)

```
3  def sieve(n):
4      "Return a list of all prime numbers less than n"
5
6      worksheet = [None, None] + list(range(2,n))
7
8      for k in range(2, ceil(sqrt(n))):
9          if worksheet[k] is not None:
10             sift(k, worksheet)
11
12     return non_nulls(worksheet)
```

Why not:

```
for k in range(2, n):
    sift(k, worksheet)
```

- No need to loop from **2** to **(n-1)**. Just need to loop from **2** to **sqrt(n)**

  Explanation is on p.87(PDF)/73(book).

- Also, operations such as

  `sift(4, worksheet)` and **`sift(8, worksheet)`**

  are unnecessary. Because **`sift(2, worksheet)`** previously would already have crossed out (i.e. mark as **None**) all the multiples of **4** and **8** as well.

  Hence, we check if **worksheet[k]** is **None** first. If it's already **None**, there is no need to call **sift(k, worksheet)**:

```
if worksheet[k] is not None:
    sift(k, worksheet)
```

- Point here is: don't loop unnecessarily --> waste time.

Running a Program from Command Line

- So far, we have been using IDLE to open/run a **.py** file.
- We will now learn another way: run a **.py** file from terminal window
- First, make sure that you can start python from your terminal
  - Windows: START --> type "**cmd**" & ENTER.
  - MacOSX: Use Finder to search for "terminal"
- Type "**python**" (Win) or "**python3**" (Mac) in your terminal window. You should see the python prompt:

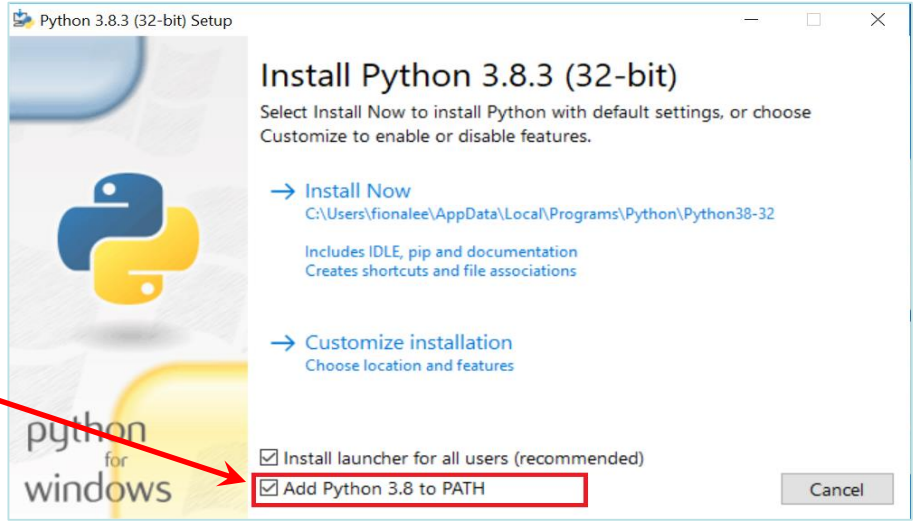
- If you see an error, that means that your PATH hasn't been correctly set

```
Microsoft Windows [Version 10.0.16299.1992]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\fionalee>python
'python' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\fionalee>_
```

- The fast way to correct that problem is to quickly uninstall Python and install it again. This time, check the "Add Python to PATH" checkbox:



Select this checkbox

… and open a NEW terminal window to try again (don't use the old terminal window)

- Create a **.py** file that contains some simple Python commands. E.g.:

test.py

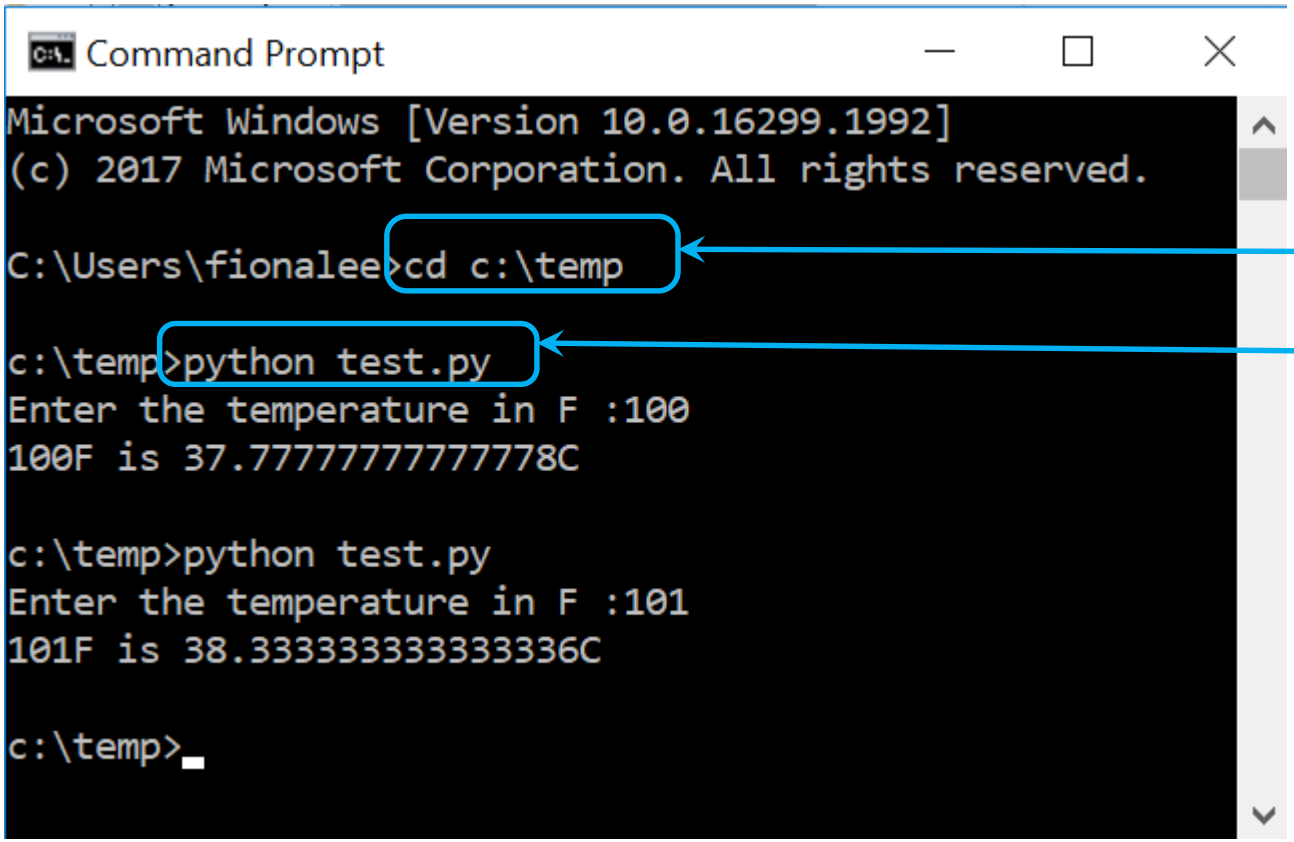
```
1  # functions
2  def celsius(f):
3      "Convert temperature f from Fahrenheit to Celsius"
4      return (f - 32) * 5 / 9
5
6
7  # main code
8  user_input = input("Enter the temperature in F :")
9
10 temp_in_f = float(user_input)
11 temp_in_c = celsius(temp_in_f)
12
13 print(user_input + "F is " + str(temp_in_c) + "C")
```

- Save it in a specific working folder (e.g. **c:\temp**) as **<filename>.py**.
- Mac users:
  You can right-click on your **py** file and select "Get Info" to see the actual folder that file is at.

- Open a new terminal window.
- Use the **cd** (Change Directory) command to go to your working folder. Use double quotation marks to enclose your working folder if there are spaces in your folder name. e.g.:

  cd "c:\comp thinking"

- Run your python program.



Change directory to your working folder

Run your py file

Lab Submission Reminder

Lab	Coverage	Release
0	Conditionals/loops	Week 1 Monday – 1 week to attempt
1	Converting pseudocode to Python code	Week 2 Monday – 1 week to attempt
2	Lists (binary search)	Week 3 Monday – 2 weeks to attempt
3	Recursion	Week 5 Monday – 1 week to attempt
4	Putting it all together	Week 6 Monday – 1 week to attempt

Contact Details

Contact your instructor anytime for:

- Help on lab
- Enquiries about projects
- Anything related to programming

Section	Instructor	Contact
G5 Thu 12:00 - 15:15	MOK Heng Ngee	hnmok@smu.edu.sg http://t.me/mokkie
G6 Wed 815 - 1130	LEE Fiona	fionalee@smu.edu.sg http://t.me/fionaleeyy
G7 Fri 815 - 1130	LEE Fiona	fionalee@smu.edu.sg http://t.me/fionaleeyy

Road Map

Algorithm Design and Analysis

- Week 1: Intro, Counting
- Week 2: Python
- Next week → Week 3: Complexity
- Week 4: Iteration & Decomposition
- Week 5: Recursion

Fundamental Data Structures

(Weeks 6 - 10)

Computational Intractability and Heuristic Reasoning

(Weeks 11 - 13)