DCS 1500:
Computational Methods

http://mtirfan.com/DCS-1500

Mohammad T. Irfan

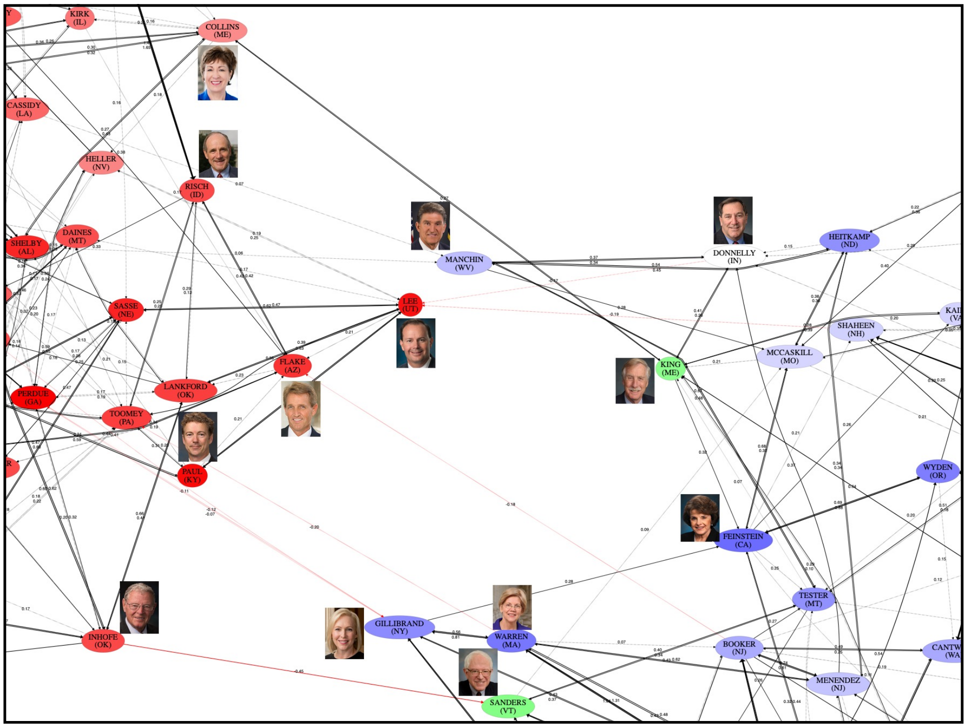
Email: mirfan@bowdoin.edu

Office Hours:
Wed 3-5:30pm, Fri 10am-12pm

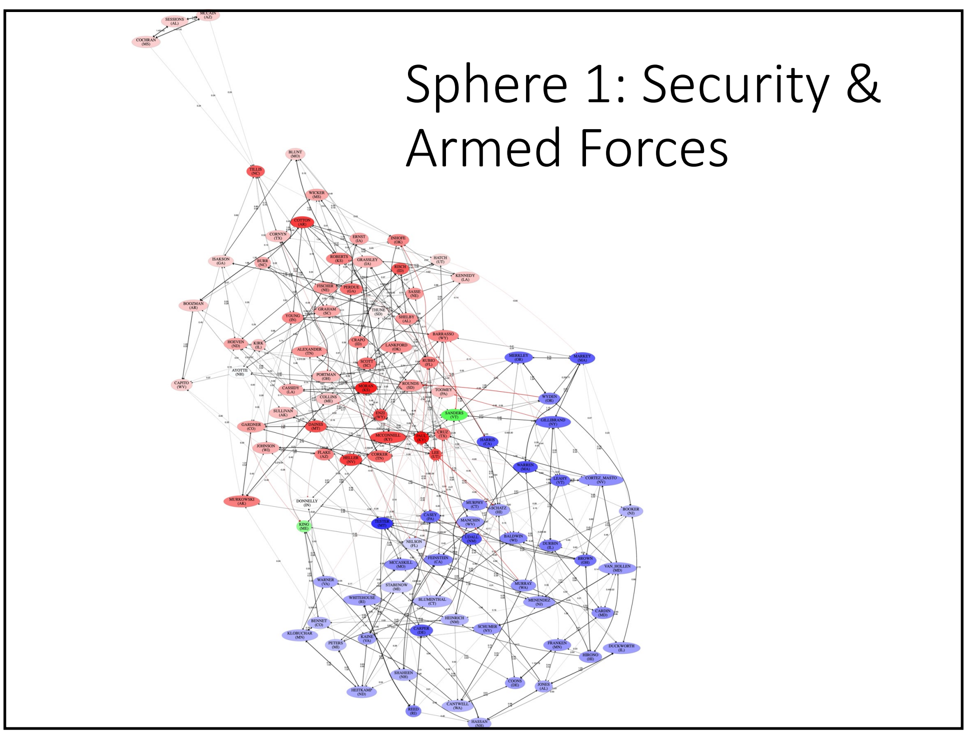
1

About myself

3

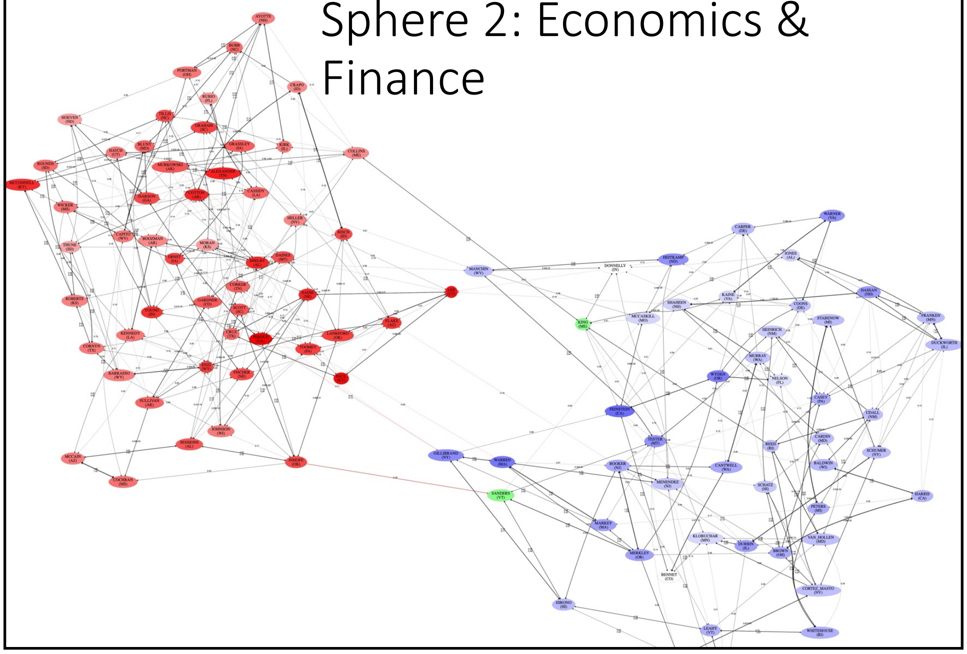

KIRK (IL)
COLLINS (ME)
CASSIDY (LA)
HELLER (NV)
RISCH (ID)
SHELBY (AL)
DAINES (MT)
SASSE (NE)
LEE (UT)
MANCHIN (WV)
DONNELLY (IN)
HEITKAMP (ND)
SHAHEEN (NH)
MCCASKILL (MO)
KING (ME)
FLAKE (AZ)
LANKFORD (OK)
PERDUE (GA)
TOOMEY (PA)
PAUL (KY)
WYDEN (OR)
FEINSTEIN (CA)
TESTER (MT)
INHOFE (OK)
GILLIBRAND (NY)
WARREN (MA)
BOOKER (NJ)
MENENDEZ (NJ)
SANDERS (VT)

4


Sphere 1: Security & Armed Forces

5

Sphere 2: Economics & Finance



6

Bowdoin Effort Earns Top Award at International Computer Science Conference Archives

July 31, 2018 by Tom Porter

Best Paper Award
AAMAS 2018

Professor Mohammad Irfan, in the middle, receives the Best Paper Award from AAMAS Program Chairs Gita Sukthankar (L) and Mehdi Dastani (R).

A research paper coauthored by a Bowdoin professor and one of his former students has earned the top spot at a recent computer science conference in Sweden. The paper employs computational game theory to model and predict congressional voting patterns. It was written by Assistant Professor of Digital and Computational Studies and Computer Science Mohammad Irfan and Tucker Gordon '17,

8

How Does Our Social Network Influence Our Behavioral Choices?

"No man is an island" wrote the poet John Donne in 1624, meaning whether we like it or not, we are all connected. It's an assertion that rings truer than ever in today's networked world, and a it's a central theme of the research currently being done by computer scientist Mohammad Irfan and his colleagues.

NSF Core Research Grant

...ssor of Digital and Computational ...and Computer Science (CS) Irfan ...d to secure around half a million ...ng for an exciting multiyear ...ing human interactions in networks. The research could have implications for many fields, he says, from public health to energy pricing to finance to the analysis of congressional voting patterns.

The award was made by the National Science Foundation (NSF) and done in collaboration with Luis E. Ortiz of the University of Michigan—Dearborn, for a multiyear research initiative. It's all part of a core NSF program called Information and Intelligent Systems, says Irfan, who is the project director (while Bowdoin is the lead organization.)

9

Published April 29, 2020 by Rebecca Goldfine

Contagion Class Turns Out to Be Prescient

Last summer, when Mohammad Irfan began planning for his new digital and computational studies class, Contagion, he had no inkling of just how relevant the subject matter would become.

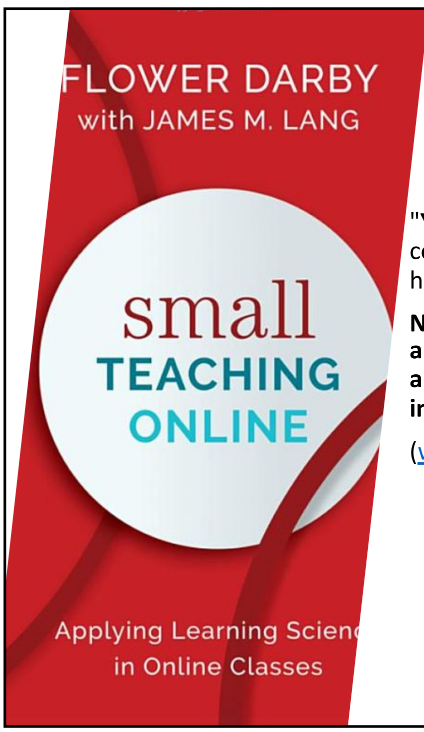
Contagion Course Spring 2020

Assistant Professor of Digital and Computational Studies and Computer Science Mohammad Irfan.

10

He was very satisfied with his teaching career.
He said something that reminded me of my own students.
He said, “The students of Collegiate School wanted to learn beyond textbooks. And if the students are not interested in learning, then there’s no joy in teaching.”
What a beautiful thing to say! That’s why I’m itching to get into a class (even during my sabbatical).

11



Advice to faculty

"**Your students want you.** Great content and a well-organized class help. But mostly they want you …

No amount of sophisticated bells and whistles can replace an authentic, present and engaged instructor."

(www.insidehighered.com)

13

I teach humans
intellectually challenging courses with
care, compassion, and emotional engagement.

14

You

15

Lunch?

Most days work for me

(MTW after 1:05 and RF ~12pm)

16

Media

1. Course website for syllabus, slides, etc.

http://mtirfan.com/DCS-1500

2. Canvas for projects and other deliverables, except Python labs

3. Coderunner for Python labs

17

Python programming environment

Download and install

- Anaconda distribution
- VS Code

Small programs: Python shell (pre-installed)

19

Student hours

- Emily Simons: Tue & Wed 7-9pm in Mills 105
- Narmer Bazile: Thu 7-9pm in Mills 105

- My office hours: Wed 3-5:30pm in Mills 209
  Fri 10am-12pm in Mills 209

23

Expressions

- Arithmetic (replace x and y by two numbers)
  - x + y
  - x – y
  - x * y
  - x/y
    - Special // operator means integer division. Example: 2//3 is 0 and 4//3 is 1.
  - x % y: remainder of the division x/y
  - x ** y: x raised to the power y
  - comparison: == (equal), != (not eq), >, >=, <, <=
- Precedence: usual
- Logical expressions: hold for now

24

Data/objects: Nouns

- Scalar – 4 types
  - int (example: -10000, 200, 53)
  - float (example: -37.59, 28.0)
  - bool (example: True, False)
  - None
- Non-scalar
  - String (example: “hello”, “57”)
  - List (example: [2, 3, 5, 7, 11, “primes”])
  - And many other ... (You can define and create your own non-scalar objects)

25

string (or text)

"String" must be in quotations (single/double)

- String concatenation

```
>>> 'Alice' + ' ' + 'Bob'
'Alice Bob'
```

- String replication

```
>>> 'Alice' * 3
'AliceAliceAlice
```

26

Statements

- `print('Welcome to Python')`
- `print("What's your name?")`
- `print (2 + 3 * 4)`
- `x = 5 * 10`

Assignment statement:
Assign the value of 5 * 10 to "variable" x

- `print(x)`

27

Variable

Simplistic definition: Names storage space

```
name = "Alice"
age = 10
```

Assignment statements are usually the only way to change the value of a variable, also the #1 source of "bugs"

28

Naming a variable

- Name must start with a letter (upper or lower case) or _ and may contain digits after the first symbol
- Use sensible names
- Cannot use reserved words (e.g., if, for, while)

29

Check in

- How does Python categorize all possible data?
- Does the following line change the value of **`age`**?
  `>>> age + 10`
- How can we change the value of a variable?

30

Syntactic error

- 1 + 2 *
- "5" + 10
- Can you suggest some more syntactic errors?

31

Comments

- # Single line comment
- '''
  Multiple lines
  of
  comments
  '''

This is basically a multiline "string" with no effect.

32

Computer programs

A mix of three types of statements

- Sequential
- Conditional (if)
- Iterative (loop)

33

First program

```
# This program asks for name and age

print('What is your name?') # ask for the name
my_name = input()
print('It is good to meet you, ' + my_name)
print('The length of your name is:')
print(len(my_name))

print('What is your age?') # ask for the age
my_age = input()
print("You'll be " + (my_age+1) + " next year.")
```

What error(s) do you see?
How to correct?

34

Some built-in functions

- print(…)
- input() #returns an input <u>string</u>
- len("some string") #returns number of characters
- int("5") #returns number 5
- str(5) #returns "5" (a string)

35

Brief Intro: Function

Reading: Ch 3
(up to "Return Values and Return Statements")

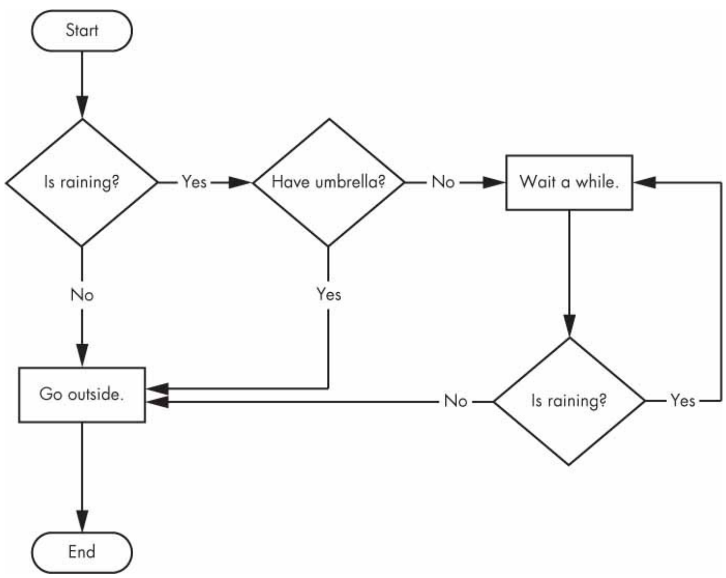
36

Flow Control

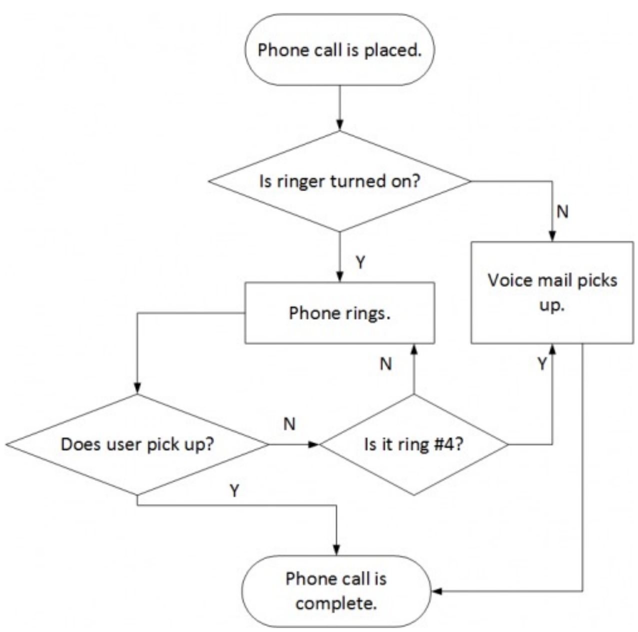
Reading: Chapter 2 of Automate the Boring Stuff

38

Flow Chart

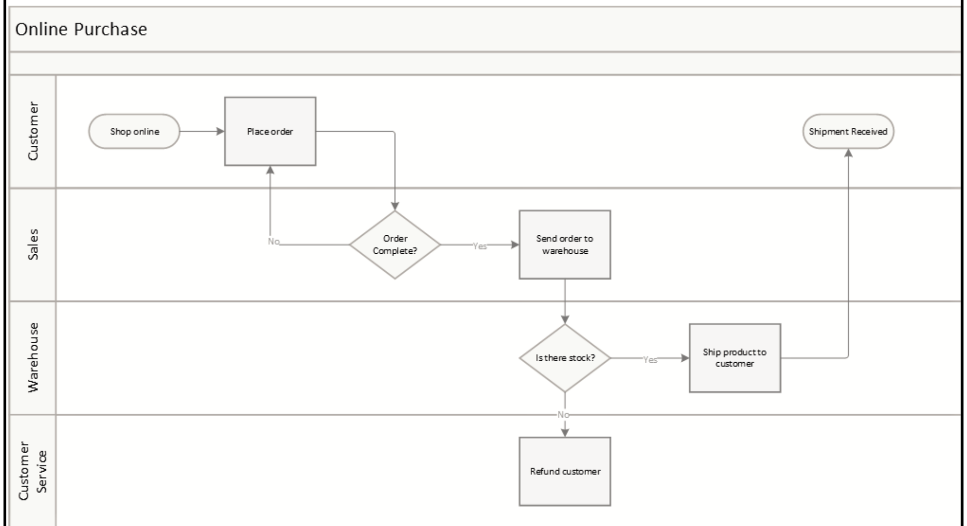


39



https://uwaterloo.ca/ist-project-management-office/tools-and-templates/tools/process-and-data-modelling-tools/process-flow-diagrams

40



https://uwaterloo.ca/ist-project-management-office/tools-and-templates/tools/process-and-data-modelling-tools/process-flow-diagrams

41

Next topics

- Boolean data
- Comparison operators
- Boolean operators
  - Truth tables
- Conditional statements (if-elif-else)
- Lots of examples

42

Boolean data

- True and False
- Examples

43

Operations with Boolean output

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

How is == different from =?

44

Boolean operators

- not – highest precedence
- and
- or – lowest precedence

```
>>> 5 > 1 or 1 + 2 == 4 and not 2 <= 3
True

>>> (5 > 1 or 1 + 2 == 4) and not 2 <= 3
False
```

45

not: truth table

Expression	Evaluates to . . .
not True	False
not False	True

46

and: truth table

Expression	Evaluates to . . .
`True and True`	`True`
`True and False`	`False`
`False and True`	`False`
`False and False`	`False`

47

or: truth table

Expression	Evaluates to . . .
`True or True`	`True`
`True or False`	`True`
`False or True`	`True`
`False or False`	`False`

48

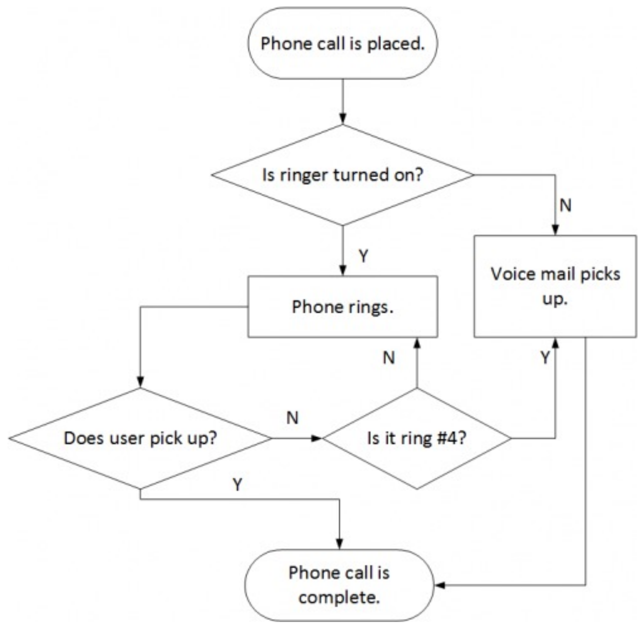
Conditional statements (if-elif-else): Syntax/grammar

```
(1) if condition:
        Block of statement(s)

(2) if condition:
        Block of statement(s)
    else:
        Block of statement(s)

(3) if condition:
        Block of statement(s)
    elif condition:
        Block of statement(s)
    …
    elif condition:
        Block of statement(s)
    else:
        Block of statement(s)
```

50

Iterative Statements or Loops

51



https://uwaterloo.ca/ist-project-management-office/tools-and-templates/tools/process-and-data-modelling-tools/process-flow-diagrams

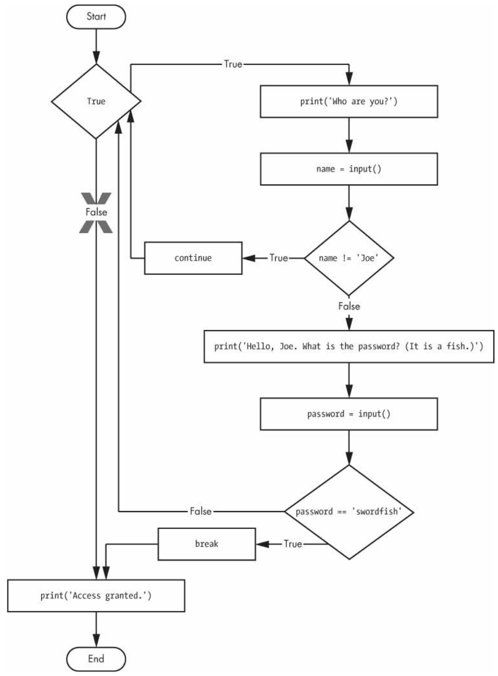
52

`while` loop

```
while condition:
    Block of statement(s)
```

Examples

53



54

for loop

```
for loop_variable in sequence:
    Block of statement(s)
```

Examples

55

range() function

To generates a sequence:

range([start = 0], end, [increment = 1])

Examples

- range (5) # 0, 1, 2, 3, 4
- range (0, 5) # 0, 1, 2, 3, 4
- range (0, 5, 1) # 0, 1, 2, 3, 4
- range(10, 70, 20) # 10, 30, 50

56

Modules

What is it? How to “import” one?

Examples

57

Functions

Chapter 3

58

Topics

- Basics of a functions: name, parameter vs argument, docstring, body of function, calling a function
- return statement
- Scopes: local vs global
- Name resolution: same name
- Call stack

59

Lists

Chapter 4

60

Topics

- Defining lists
  - Heterogenous data
- Revisit len() function
- Concatenation
- Methods: append, insert, remove, pop, sort, reverse

61

List

- Ordered sequence of values
- Each value is identified by an index
- Examples
  - `heart_rates = [98, 75, 80, 90]`
  - `times = ["16:00", "16:05", "16:10", "16:15"]`

62

More examples of list

A list may contain heterogeneous data

```
list1 = ["I did it all", 4, "love"]
```

63

len(...) function

returns the length/number of elements

```
heart_rates = [98, 75, 80, 90]
print(len(heart_rates))  # 4
```

65

Indexing and slicing

```
v = [20, 30.5, 5.1, 10.2, 100]
```

- Indexing
  - `v[2]        # 5.1`
  - `v[-1]       # 100`
  - `v[-2]       # 10.2`
- Slicing (stops short of the end index)
  - `v[1:3]      # [30.5, 5.1]`
  - `v[0:len(v)] # [20, 30.5, 5.1, 10.2, 100]`
  - `v[0: -2]    # [20, 30.5, 5.1]`
  - `v[:3]       # [20, 30.5, 5.1]`
  - `v[2:]       # [5.1, 10.2, 100]`
  - `v[:]        # [20, 30.5, 5.1, 10.2, 100]`

66

Concatenation

Join multiple lists by +

```
x = [10, 20]
y = [5, 30, 10]
z = x + y
print(z) # [10, 20, 5, 30, 10]
```

67

Modifying a list

Lists are “mutable”

```
x = [5, 10, 15]
```

Add

#append an object e to x: x.append(e)
`x.append(20)` # x is now [5, 10, 15, 20]
#To insert object e at index i: x.insert(i,e)
`x.insert(1, 7)` # x is now [5, 7, 10, 15, 20]

Delete

#remove the first occurrence of an object
`x.remove(15)` # x is now [5, 7, 10, 20]
#remove and **return** the item at a given #index i: x.pop(i)
`y = x.pop(2)` # x = [5, 7, 20], y = 10

68

More list "methods"

- #sort a list (low to high)
  `x.sort()` # x will be changed (in sorted order)
  - Common mistake: `x = x.sort()`
  - Above, x = None because sort function returns None
- #reverse a list
  `x.reverse()` # x will be changed (reversed)
  - Common mistake: `x = x.reverse()`

69

for loops and lists: a great match!

- **Coding problem:** Write a function that takes a list of heart rates as a parameter and returns the average heart rate.

- **Coding problem:** Write a function that takes a list of heart rates as a parameter and returns the maximum heart rate.

- **Coding problem:** Write a function *min_end(nums)* that takes a list of ints *nums*. It figures out which of the first and last elements in the list is the smaller and sets all the other elements of *nums* to be that value. Return the changed list *nums*. (Q11 in Lab 5)

70

Strings

Chapter 6

71

String

- Examples
  - `"Hello and welcome"`
  - `"Bowdoin College"`
- Multiline string (where did I see this before?)

```
print('''Dear Alice,

Eve's cat has been arrested for
catnapping, cat burglary, and
extortion.

Sincerely,
Bob''')
```

- Assignment statement
  - `college = "Bowdoin College"`

72

Operations on String

- `college = "Bowdoin College"`
- Length of a string
  - `len(college)     # 15`
- Indexing
  - `college[2]       # 'w'`
  - `college[-1]      # 'e'`
  - `college[-2]      # 'g'`
- Slicing
  - `college[0:2]     # 'Bo'`
  - `college[0:len(college)]`
    `                 # 'Bowdoin College’`
  - `college[ : : 3] # 'Bdnoe'`

73

in and not in operators

```
>>> 'Hello' in 'Hello, World'
True

>>> 'cats' not in 'cats and dogs'
False
```

74

Concatenation

Use the + operator to join strings

```
college = "Bowdoin College"
city = "Brunswick, ME"
print(college + city) #Bowdoin CollegeBrunswick, ME
print(college + " " + city) #Bowdoin College Brunswick, ME
print(college + "\n" + city)
#Bowdoin College
#Brunswick, ME
```

75

Whitespace characters

- " " → space
- "\n" → new line
- "\t" → tab

76

Put one string in another

```
>>> name = 'Al'
>>> age = 4000
>>> 'Hello, my name is ' + name + '. I am ' +
str(age) + ' years old.'
'Hello, my name is Al. I am 4000 years old.'
```

Alternative:

```
>>> name = 'Al'
>>> age = 4000
>>> 'My name is %s. I am %s years old.' % (name,
age)
'My name is Al. I am 4000 years old.'
```

77

Splitting a string

- Breaks up a string into parts by a separator
- “returns” a list of parts

```
college = "Bowdoin College"
#split by whitespace
parts_list = college.split()
print(parts_list) #['Bowdoin', 'College']
```

- Splitting doesn't change the original string

```
print(college) #Bowdoin College
```

78

Splitting a string (cont...)

- You can choose any separator

```
friends_str = "Allen, Emma, Bob, Cindy"
friends_list = friends_str.split(", ")
print(friends_list)
#['Allen', 'Emma', 'Bob', 'Cindy']
```

Space

79

Other string methods

- `upper(), lower(), isupper(), islower(), isalpha(), isalnum()`
- Example:

```
college = college.upper()
```

80

Other string methods

```
>>> 'Hello, world!'.startswith('Hello')
True

>>> 'Hello, world!'.endswith('world!')
True
```

81

List vs. string

- Strings are "immutable"
  - Cannot do:
    st = "hello"
    st[0] = "H"
  - Can do:
    st = "hello"
    st = "Hello"
- These functions are only for lists, not for strings:
  - append, insert, remove, pop, sort, reverse, ...

82

Taking a string as input

- `x = input ("Enter your name: ")`
- `y = input ("How old are you? ")`
  - Suppose user enters 57 as his/her age
  - y's value is "57", not 57
- if you want to convert string "57" to number 57
  - `z = int(y)`
- int(y) converts a compatible string y to int
- str(z) converts a number z to string

83

What is the difference between "57" and 57?

- Main difference is representation
- You can apply all arithmetic operators on 57
  - `57 + 2`
  - `57 * 2`
- Arithmetic operations are meaningless for "57"
  - `"57" + 2                    # ERROR`
  - `"57" * 2                    # '5757'`

84

Python 3 built-in functions
https://docs.python.org/3/library/functions.html

2. Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

85

Dictionaries and structured data

86

What is it?

- Dictionary maps "keys" to "values"
  - Keys are like indices of list, only that they don't need to be numbers
- Keys
  - Immutable objects
  - strings, numbers, and "tuples" of immutable objects
- Values
  - Any object

87

How to create a dictionary

- Example: map people to their home states

```
home = {} #Empty dictionary
home["Cindy"] = "ME" #Enters a new mapping ("Cindy" : "ME")
home["Alice"] = "MA" #Maps "Alice" to "MA"
home["David"] = "NY"
home["Bob"] = "NY"
print(home) #Ordering: random in Python 3.5, sequential in 3.6
```

```
{'David': 'NY', 'Cindy': 'ME', 'Bob': 'NY', 'Alice': 'MA'}
```

- Another way

```
home = {'David': 'NY', 'Cindy': 'ME', 'Bob': 'NY', 'Alice': 'MA'}
print(home)
```

```
{'Alice': 'MA', 'David': 'NY', 'Bob': 'NY', 'Cindy': 'ME'}
```

88

Typical operation 1: Given a key, find the value

- What is the home state of Alice?

```
print(home["Alice"])
```

```
MA
```

Note the brackets

- Harder question: given a value, find the key(s)
  - Class participation

89

Typical operation 2: Change the value of a key

- Alice has moved from MA to NY…

```
home["Alice"] = "NY" #Alice's home state is changed to NY
print(home) #Verify it
```

```
{'Alice': 'NY', 'David': 'NY', 'Bob': 'NY', 'Cindy': 'ME'}
```

90

Typical operation 3: in operator– presence of a key

- Check if a key is present

```
if "David" in home:
    print("David -->", home["David"])
if "Estie" in home:
    print("Estie -->", home["Estie"])
```

```
David --> NY
```

- Iterate over all keys

```
for person in home:
    print(person, "-->", home[person])
```

```
Alice --> NY
David --> NY
Bob --> NY
Cindy --> ME
```

91

keys() method to get a seq. of keys

```
people = home.keys()
print(people)
```

```
dict_keys(['Alice', 'David', 'Bob', 'Cindy'])
```

Also, check out the values() and items() methods

92

Sorting

Sort a dictionary by keys

```
names = sorted(home)
```

93

Sorting

```
#Sort a dictionary by values
life_span = {"dog": 8, "cat": 12, "fox": 7, "horse": 15}
names_sorted = sorted(life_span, key = life_span.get, reverse = True)
print(names_sorted)

#Print names and life spans in sorted order
for name in names_sorted:
    print(name, "-->", life_span[name])
```

```
['horse', 'cat', 'dog', 'fox']
horse --> 15
cat --> 12
dog --> 8
fox --> 7
```

94

Files

Chapter 9

95

Paths

- Directory structure in computer: tree
- Commands in terminal
  - pwd
  - ls
  - cd *path*
    - **..** for *path* means parent folder, **../..** means two levels up
    - Relative path: begins with the current directory
    - Absolute path: begins with the "root" folder
- More here: https://www.techrepublic.com/article/16-terminal-commands-every-user-should-know/

96

Read a file – 3 steps

1. Create a file object
   - **file_object = open(file_name, "rt")**
   - file_name is a string
   - Concept of file path used here
   - “rt” means read text. It’s the mode of file operation.

2. Read using the file object
   - **content = file_object.read()**
     #reads the whole content into a string called content

3. Close the file object
   - **file_object.close()**

97

Alternative: readlines()

```
lines = file_object.readlines()
```

lines is a list of strings– each string is a line, which may contain a trailing newline character

98

Write to a file – 3 steps

1. Create a file object
   - **file_object = open(file_name, "wt")**
   - file_name is a string
   - "wt" means write text. It's the mode of file operation.
2. Write to the file object
   - **file_object.write(content)** #write a string to file
3. Close the file object
   - **file_object.close()**

99

Review: string → list

- Want a list: every single line of the string becomes an individual item of the list
- list_lines = st.split("\n") New line character
  - list_lines is the name of the list
  - st is the name of the string

100

List → String

- Want a string where every item of the list becomes an individual line of the string
- st = "\n".join(list_lines)

101

f-strings

f-strings have an `f` prefix before the starting quotation mark allow variables to be placed inside braces { }

```
>>> name = 'Al'
>>> age = 4000
>>> f'My name is {name}. Next year I
will be {age + 1}.'
'My name is Al. Next year I will be
4001.'
```

102

Problem: Count the number of words, lines, and characters in a file

Save the stats in another file

103

```
def file_stat(file_name):
    """Count the number of words, lines, and characters in a text file
    Save the info in a new file"""
    #Read the file first
    #1. Create file object
    file_object = open(file_name, "rt")
    #2. Read the file
    content = file_object.read()
    #3. Close the file object
    file_object.close()

    #Count the number of words
    word_list = content.split()
    line_list = content.split("\n")

    print(f"Number of words: {len(word_list)}")
    print(f"Number of lines: {len(line_list)}")
    print(f"Number of characters: {len(content)}")

    output = f"{len(word_list)}\n{len(line_list)}\n{len(content)}"
    file_object = open("stat.txt", "wt")
    file_object.write(output)
    file_object.close()

file_stat("files/rainyday.txt")
```

Change it according to where you saved the file

104

More on text files

- Examples: textfiles.com
- Text file vs. Word document
- Different encodings for text files
  - Mac, Unix, Windows
  - New-line character
    - Unix: always "\n"
    - Mac: before 2001 (OS 9) "\r"; OS X "\n"
    - Windows: "\r\n"
    - Excel tab delimited file: "\r"
    - Python (e.g., readlines() function) can handle all!

105

`with:` better alternative for file operations

- Cleaner code
- file is closed properly even if there's an error or exception

- Reading

```
with open('input.txt', 'rt') as file:
     content = file.read()
```

- Writing

```
with open('output.txt', 'wt') as file:
     file.write('Hello world')
```

116

Working with Fitbit data

Problem: Delete every line where the heart rate is 0

117

Working with CSV files

Ch 16

118

Working with JSON files

Ch 16

119

Working with APIs

Ch 16

120

Conditional Statements

Write each piece of codes in a separate cell in Jupyter Notebook

151

if-else

- An if block can be *optionally* associated with an else block
- Python ignores the else block when the if condition is true

```
steps_done = 9000
if steps_done >= 10000:
    print("Bravo! You've done", steps_done, "steps!")
else:
    print("You need to do more than", steps_done, "steps.")
```

Note the indentation

152

Spice it up…
Can we take steps_done as an input?

1. Takes a "string" as input

2. Converts that string into integer

```
steps_done_str = input("Enter today's number of steps: ")
steps_done = int(steps_done_str)
if steps_done >= 10000:
    print("Bravo! You've done", steps_done, "steps!")
else:
    print("You need to do more than", steps_done, "steps.")
```

153

Spice it even more…
Use a function

Comments

More on functions later on …

Function name

Parameters

```
#steps_done is the number of steps done today
#function gives feedback on whether to do more steps

def give_feedback(steps_done):

    if steps_done >= 10000:
        print ("Bravo! You've done", steps_done, "steps!")
    else:
        print ("You need to do more than", steps_done, "steps.")
```

Function header

Function body: Note the indentation

How to run/call this function?

154

Calling a function

- The function give_feedback will never work unless we call it!
- Run the following code to call this function

give_feedback(11000)

No "def" here

Argument
(steps_done will get the value of 11000)

155

Logical expressions

- Logical operators: a and b are ***bool*** type (that is, replace each of a and b by either True or False)
  - a **and** b
  - a **or** b
  - **not** a
- Examples
  - 3 > 2 and 3 > 4
  - 3 > 2 or 3 > 4
  - not 3 > 4

156

if-elif-else

- **if block**, followed by any number of **elif blocks**, followed by an optional **else block**
  - Python will execute at most one branch
- **Problem:** Given the average calories burned for 3 people, find who burned the most cal.

```
#x, y, z are the calories burned
#function announces who burned most cal.
def get_max(x, y, z):
    if x > y and x > z:
        print ("First person")
    elif y > z:
        print ("Second person")
    else:
        print ("Third person")
```

157

Quiz

- What will be the output?

```
x = 10000
if x >= 10000:
        print ("OK")
elif x <= 10000:
        print ("Not OK")
```

158

Quiz

- What will be the output?

```
x = 9000
if x >= 10000:
        print ("OK")
```

159

if statements can be *nested*

- Modify the get_max program so that it also detects ties
- Think about the logic first

160

Solution

```
#x, y, z are the calories burned
#function announces who burned most cal.
def get_max(x, y, z):
    if x > y and x > z:
        print ("First person")
    elif y > z:
        print ("Second person")

        if y == x: #See if y is equal to x
            print ("Tied with the first person")
    else:
        print ("Third person")

        if z == x:
            print ("Tied with the first person")

        if z == y: #Will elif do the job here?
            print ("Tied with the second person")
```

161

For practice: Guttag (p. 16)

- Write a program to find the largest odd number among three given numbers (not necessarily odd) x, y, and z.

```
def get_largest_odd(x, y, z):
  if x%2 == 1 and y%2 == 1 and z%2 == 1:
    if x > y and x > z:
      print x
    elif y > z:
      print y
    else:
      print z
  elif x%2 == 1 and y%2 == 1:
    if x > y:
      print x
    else:
      print y
  elif y%2 == 1 and z%2 == 1:
    if y > z:
      print y
    else:
      print z
```

```
  elif z%2 == 1 and x%2 == 1:
    if z > x:
      print z
    else:
      print x
  elif x%2 == 1:
    print x
  elif y%2 == 1:
    print y
  elif z%2 == 1:
    print z
  else:
    print "None is odd"
```

162

Data in Python

168

Data/Objects

- Scalar – 4 types
  - int (example: -10000, 200, 53)
  - float (example: -37.59, 28.0)
  - bool (example: True, False)
  - None
- Non-scalar
  - String (example: “hello”, “57”)
  - List (example: [2, 3, 5, 7, 11, “primes”])
  - And many other ... (You can define and create your own non-scalar objects)

169

Non-scalar data: List

171

Fitbit data

Date	Calories Burned	Steps	Distance	Floors	Minutes Sedentary	Minutes Lightly Active	Minutes Fairly Active	Minutes Very Active	Activity Calories
2015-09-08	1,265	0	0	0	1,440	0	0	0	0
2015-09-09	1,265	0	0	0	1,440	0	0	0	0
2015-09-10	1,744	5,807	2.31	12	1,274	166	0	0	542
2015-09-11	2,127	9,679	3.85	6	1,096	344	0	0	1,053
2015-09-12	1,852	5,747	2.29	6	1,165	275	0	0	773
2015-09-13	1,517	2,714	1.08	6	1,310	130	0	0	332
2015-09-14	1,937	7,484	2.98	24	1,170	263	4	3	850
2015-09-15	1,866	7,801	3.1	21	1,159	281	0	0	800
2015-09-16	1,813	6,256	2.49	17	1,204	236	0	0	680
2015-09-17	1,882	8,252	3.28	12	1,191	240	4	5	786
2015-09-18	1,805	5,976	2.38	14	1,097	267	0	0	734
2015-09-19	2,035	10,190	4.05	9	1,097	324	18	1	1,043
2015-09-20	1,895	7,199	2.86	14	1,148	292	0	0	854
2015-09-21	1,797	7,309	2.91	17	1,227	213	0	0	660
2015-09-22	1,265	0	0	0	1,440	0	0	0	0
2015-09-23	1,727	5,522	2.2	12	1,230	194	3	13	594
2015-09-24	1,605	5,186	2.06	12	1,375	65	0	0	224
2015-09-25	1,929	10,309	4.1	16	1,177	244	5	14	845
2015-09-26	2,129	10,702	4.26	6	1,058	368	14	0	1,165
2015-09-27	1,797	6,419	2.55	11	1,186	254	0	0	716
2015-09-28	1,964	11,177	4.44	15	1,186	189	9	56	870
2015-09-29	1,269	42	0.02	1	1,439	1	0	0	4
2015-09-30	1,834	7,096	2.82	17	1,198	232	9	1	720
2015-10-01	1,802	8,854	3.52	22	1,252	157	13	18	664
2015-10-02	1,758	5,704	2.27	7	1,206	234	0	0	654
2015-10-03	1,728	5,909	2.35	12	1,231	209	0	0	600
2015-10-04	1,571	4,380	1.74	8	1,197	137	0	0	408
2015-10-05	784	0	0	0	893	0	0	0	0

172

Date Time	HRate
9/14/15 16:00	98
9/14/15 16:05	75
9/14/15 16:10	80
9/14/15 16:15	90
9/14/15 16:20	107
9/14/15 16:25	103
9/14/15 16:30	108
9/14/15 16:35	81
9/14/15 16:40	85
9/14/15 16:45	91
9/14/15 16:50	105
9/14/15 16:55	115
9/14/15 17:00	113
9/14/15 17:05	105
9/14/15 17:10	128
9/14/15 17:15	121
9/14/15 17:20	129
9/14/15 17:25	105
9/14/15 17:30	96
9/14/15 17:35	100
9/14/15 17:40	89
9/14/15 17:45	78
9/14/15 17:50	74
9/14/15 17:55	84
9/14/15 18:00	96

173

Iterative Statements/ Loops

1. while loops – skipped here
2. for loops– mostly used in Python

174

Announcements

- Assignment 3
  - Due next week Thursday
  - Collaboration policy
- Python Quiz 2
  - Next Thursday, 11/2

175

for loop

- General form (not actual code)
  - for ***loop_variable*** in ***sequence***:

      body of for loop

- sequence
  1. Built-in sequence type **range**
  2. Any list of your own
  3. Even a string!

179

range to generate sequence

- sequence is commonly specified using ***range*** with 3 parameters:
  - start (optional, default is 0)
  - end (must specify, actually ends before this value)
  - increment (optional, default is 1)
- Examples
  - range(10, 70, 20) # generates [10, 30, 50]
  - range (0, 5, 1) # [0, 1, 2, 3, 4]
  - range (0, 5) # [0, 1, 2, 3, 4]
  - range (5) # [0, 1, 2, 3, 4]

To print the output:
print(list(range(...)))

180

for loop

- Problem: Print numbers 1, 2, …, 10, each on a single line

Loop variable

Sequence

```
def print_numbers():
  for i in range(1, 11, 1):
    print (i)
```

Body – multiple lines allowed

181

for loop (cont…)

- Problem: Square a positive integer only

```
def square(x):

    ans = 0
    for times in range(x):
        ans = ans + x
    print (ans)
```

Alternatives:
range(0, x, 1)
range (0, x)
range (1, x+1, 1)
range (1, x+1)

Can you make it work for x <= 0 as well?

182

for loop on lists

```
#Function prints heart rate data in TWO WAYS
#Parameter: heart_rates is a list of integers
def print_data(heart_rates):
    #Print all the list elements, one in each line
    print("First way:")
    for i in range(len(heart_rates)):
        print(heart_rates[i])

    #Do the same in a different way
    print("Second way:")
    for x in heart_rates:
        print(x)

#Call the function
print_data([98, 75, 80, 90])
```

```
First way:
98
75
80
90
Second way:
98
75
80
90
```

185

Problem:
find the average of a list of heart rates

186

```python
#Function calculates the average heart rate
#Parameter: heart_rates is a list of integers
def calc_avg(heart_rates):
    #Step 1. Calculate total
    total = 0
    for rate in heart_rates: #rate is actual element, not index
        total = total + rate #accumulate numbers

    #Step 2. Divide total by the number of elements
    avg = total/len(heart_rates)
    print("Average is:", avg)

#Call the function
calc_avg([98, 75, 80, 90])
```

```
Average is: 85.75
```

187

Loop and string

188

for loop over words in string

```
def show_words(st):
    for x in st.split(): # iterate over the parts/words of st
        print x    # This will print the individual words of st
```

190

Functions

return statements

Later on: Scoping rules

191

Example

- Write a function to get (not print) the maximum of two numbers

```
#return the maximum of x and y
def get_max(x, y):
    if x > y:
        return x
    else:
        return y

get_max(10, 20)
#Didn't save the returned value. It's lost.
print("The max was:", ??? )
```

This is how the built-in max(...) function is defined!

- The return value is lost unless caller saves it!

```
t = get_max (10, 20)  # t holds 20
print("The max was:", t)
```

192

Semantics of return statement

- Two types of return statement
  - *return someData* #returns someData to the caller
  - *return* #returns None to the caller;
    #implicit in any function that does not say return
- A function terminates immediately after executing a return statement
  - The returned data is transmitted to the caller

193

return vs. print

- return sends data from callee to caller.
  print just prints the data in callee, does not transmit data.
- return immediately terminates a function;
  print does not.

194

Demo: loop, function, return

```
#Parameter: k is the PIN code entered by a user
#Returns True for correct PIN, False otherwise
def validate(k):
    correct_PIN = 2289
    if k == correct_PIN:
        return True
    else:
        return False

#A "brute force" attempt at breaking a 4-digit PIN code
def break_PIN():
    for n in range(10000): #from 0 to 9999
        if validate(n):
            print("Success: PIN is", n)

break_PIN()
```

```
Success: PIN is 2289
```

195

Improved version: account for leading 0s

```
#Parameter: k is the PIN code entered by a user
#Returns True for correct PIN, False otherwise
def validate(test_PIN):
    correct_PIN = [0, 0, 8, 9]
    if test_PIN == correct_PIN:
        return True
    else:
        return False

#A "brute force" attempt at breaking a 4-digit PIN code
def break_PIN():
    for x in range(10000): #from 0 to 9999
        digit0 = x // 1000 #integer division to get the left-most digit
        rest = x % 1000 #the remaining 3 digits
        digit1 = rest // 100
        rest = rest % 100
        digit2 = rest // 10
        rest = rest % 10
        digit3 = rest

        digit_list = [digit0, digit1, digit2, digit3]

        if validate(digit_list):
            print("Success: PIN is", digit_list)

break_PIN()
```

```
Success: PIN is [0, 0, 8, 9]
```

196

Why is return important?

- Gives a way to use previously defined functions
- Makes interactions among functions possible in a large project

198

File operations

Read a file
Write to a file
Append to a file

200

Read a file – 3 steps

1. Create a file object
   - **file_object = open(file_name, <u>“rt”</u>)**
   - file_name is a string—must have the full path to the file unless the file is in the current directory.
   - “rt” means read text. It’s the mode of file operation.

2. Read the file object
   - **big_str = file_object.read()**
     #reads the whole content into a string

3. Close the file object
   - **file_object.close()**

201

Write to a file – 3 steps

1. Create a file object
   - **file_object = open(file_name, <u>“wt”</u>)**
   - file_name is a string—must have the full path to the file unless the file is in the current directory.
   - “wt” means write text. It’s the mode of file operation.

2. Write to the file object
   - **file_object.write(big_str)** #write a string to file

3. Close the file object
   - **file_object.close()**

202

Text files

- Resource: textfiles.com
- Differences between a text file and a Word document
- Different encodings for text files
  - Mac, Unix, Windows
  - New-line character
    - Unix: always "\n"
    - Mac: before 2001 (OS 9) "\r"; OS X "\n"
    - Windows: "\r\n"
    - Excel tab delimited file: "\r"
    - Python can handle all!

203

Problem: Count the number of words, lines, and characters in a file

Save the info in another file

204

```python
#This function counts the number of lines, words, and
#characters in a given file and writes these to a new file.
#Parameter: file_name is the name of file to be analyzed.
def process_file_stat(file_name):
    #Read from the file
    file_object = open(file_name, "rt")
    big_str = file_object.read()
    file_object.close()

    #Calculate
    lines = big_str.split("\n")
    words = big_str.split()

    msg = "# lines = "+ str(len(lines)) + "\n" + \
        "# words = " + str(len(words)) + "\n" + \
        "# characters = " + str(len(big_str))
    print(msg)

    #Save the info to a new file
    file_object = open("info.txt", "wt")
    file_object.write(msg)
    file_object.close()

#Remember to call this function!
```

205

Text processing using strings

Review len, indexing, slicing, concat.

Help: https://docs.python.org/3/library/stdtypes.html#string-methods

211

Text manipulation

- String functions/*__methods__*
  - String is an "object" (non-scalar data)
  - In general, for objects there are predefined functions:

objectName.methodName(parameters)

- **Original string is *never* modified!**
  In *most cases*, string methods return a modified copy of the original string.

213

String methods

- In the following methods, st is a predefined string
- resultSt = st.capitalize()
- resultSt = st.upper()
- resultSt = st.lower()
- resultSt = st.title()
- resultSt = st.swapcase()
- resultTF = st.startswith(*anotherString*)
- resultTF = st.endswith(*anotherString*)
- index = st.find(searchStr [, startIndex, endIndex])
- resultSt = st.replace(searchStr, replaceStr [, count])
- resultTF = st.isalpha()
- resultTF = st.isdigit()
- resultList = st.split([delimiter])

Replaces all matching if count is not given

Default delimiter: white space

214

String → List

- Want a list: every single line of the string becomes an individua~~l item o~~ New line character
- list_lines = st.split("\n")
  - list_lines is the name of the list
  - st is the name of the string

216

List → String

- Want a string s.t. every item of the list becomes an individual line of the string
- st = “\n”.join(list_lines)

217

Python Assignment 4
Practice Problem on Fitbit: Delete every line where the heart rate is 0

New-line character: '\n

Tab character: '\t'

219

Heart rate data file (snapshot)

9/10/15 10:00	0
9/10/15 10:05	0
9/10/15 10:10	0
9/10/15 10:15	0
9/10/15 10:20	0
9/10/15 10:25	82
9/10/15 10:30	76
9/10/15 10:35	84
9/10/15 10:40	84
9/10/15 10:45	85
9/10/15 10:50	91
9/10/15 10:55	99
9/10/15 11:00	89

220

Algorithm

- How to get the input?
  - Design question: How to get the file name?
  - Read the data file into a string (big_str)
- How to store/save the output?
  - Use a list to incrementally store the output
  - Create an empty list of desired_lines, which we'll populate later
- How to incrementally populate desired_lines?
  - Split the big_str into lines
  - For each line do:
    - Check if the line has a heart rate of 0. If not, append the line to the list of desired_lines
- How to save the output to a file?
  - Convert the desired_lines list to a string
  - Write that string to a file
  - Design question: Should we replace the old file or create a new file?

9/10/15 10:00	0
9/10/15 10:05	0
9/10/15 10:10	0
9/10/15 10:15	0
9/10/15 10:20	0
9/10/15 10:25	82
9/10/15 10:30	76
9/10/15 10:35	84
9/10/15 10:40	84
9/10/15 10:45	85
9/10/15 10:50	91
9/10/15 10:55	99
9/10/15 11:00	89

221

```
#This function takes the file name of heart rates data as a parameter.
#It's job is to delete all the lines where the heart rate is 0 and
#   save the modified content to a new file.
def delete_zeros(file_name):
    #3 steps of reading a file
    file_object = open(file_name, "rt")
    big_str = file_object.read()
    file_object.close()

    #Create an empty list to incrementally save output
    desired_lines = []

    #Incrementally populate desired_lines
    original_lines = big_str.split("\n")
    for line in original_lines:
        if not line.endswith("\t0"):
            desired_lines.append(line)

    #Save desired_lines to a file
    #First, convert the list to a string
    output_str = "\n".join(desired_lines)

    #Now, write the string to a file
    file_name = "new_" + file_name #prepends "new_" to prev file_name
    file_object = open(file_name, "wt")
    file_object.write(output_str)
    file_object.close()

#Call the function
delete_zeros("heart_p2_partial_2.txt")
```

222

Problem: Replace all occurrence of a string in a file by another string

What are the steps?

223

Additional topics in Python

- Dictionary
- break and continue
- Modules
- Scoping rules of functions

228

Data Structure
Dictionary

229

What is it?

- Dictionary maps "keys" to "values"
  - Keys are like indices of list, only that they don't need to be numbers
- Keys
  - Immutable objects
  - strings, numbers, and "tuples" of immutable objects
- Values
  - Any object

230

How to create a dictionary

```
home = {} #Empty dictionary
home["Cindy"] = "ME" #Enters a new mapping ("Cindy" : "ME")
home["Alice"] = "MA" #Maps "Alice" to "MA"
home["David"] = "NY"
home["Bob"] = "NY"
print(home) #Ordering: random in Python 3.5, sequential in 3.6
```

```
{'David': 'NY', 'Cindy': 'ME', 'Bob': 'NY', 'Alice': 'MA'}
```

```
home = {'David': 'NY', 'Cindy': 'ME', 'Bob': 'NY', 'Alice': 'MA'}
print(home)
```

```
{'Alice': 'MA', 'David': 'NY', 'Bob': 'NY', 'Cindy': 'ME'}
```

231

Typical operation 1: Given a key, find the value

- What is the home state of Alice?

```
print(home["Alice"])
```

```
MA
```

Note the brackets

- Harder question: given a value, find the key(s)
  - Class participation

232

Typical operation 2:
Change the value of a key

- Alice has moved from MA to NY...

```
home["Alice"] = "NY" #Alice's home state is changed to NY
print(home) #Verify it
```

```
{'Alice': 'NY', 'David': 'NY', 'Bob': 'NY', 'Cindy': 'ME'}
```

233

Typical operation 3:
in operator– presence of a key

- Check if a key is present

```
if "David" in home:
    print("David -->", home["David"])
if "Estie" in home:
    print("Estie -->", home["Estie"])
```

```
David --> NY
```

- Iterate over all keys

```
for person in home:
    print(person, "-->", home[person])
```

```
Alice --> NY
David --> NY
Bob --> NY
Cindy --> ME
```

234

keys() method to get a seq. of keys

```
people = home.keys()
print(people)
```

```
dict_keys(['Alice', 'David', 'Bob', 'Cindy'])
```

235

Email project

Find the most active weekday for email exchange

- Outline:
- Start with a dictionary like

```
frequency = {"Mon":0, "Tue":0, "Wed":0,
                 "Thu":0, "Fri": 0, "Sat":0,
"Sun":0}
```

- For each email, get the weekday
  - Increase the frequency of that weekday by 1
    `frequency[weekday] += 1`
- Find the weekday that has the highest frequency

236

Email project

Find the top 5 most frequent domain names
→ Sort a dictionary by values

- Similar problem: Consider a dictionary of animal names and life spans. Sort the names by life spans high to low.
- Built-in function:
  - sorted(name of dictionary, key = sort by what?, low to high or high to low?)

237

```
#Sort a dictionary by values
life_span = {"dog": 8, "cat": 12, "fox": 7, "horse": 15}
names_sorted = sorted(life_span, key = life_span.get, reverse = True)
print(names_sorted)

#Print names and life spans in sorted order
for name in names_sorted:
    print(name, "-->", life_span[name])
```

```
['horse', 'cat', 'dog', 'fox']
horse --> 15
cat --> 12
dog --> 8
fox --> 7
```

238

break
continue

Both apply to loops only

(not to if statements)

239

break statement

- Terminates a loop immediately
  - Rest of the code after the loop will be executed
  - The function is <u>not</u> terminated (contrast: return)
- Problem: Given a list of heart rates, check if there's any rate > 100 and print a "found" message in that case

```
def check(heart_rates):
    for x in heart_rates:
        if x > 100:
            print("Found:", x)
            break #terminates the for loop
    print("Done") #Check out: break doesn't terminate function

check([60, 70, 110, 65, 120, 80, 70, 115, 130])
```

```
Found: 110
Done
```

240

continue statement

- Goes to the next iteration of the loop by skipping the rest of the code inside the loop after the continue statement
- Problem: Given a list of heart rates, print all non-zero heart rates

```
def print_nonzero(heart_rates):
    for x in heart_rates:
        if x == 0:
            #skip the rest of the loop & go to the next iteration
            continue
        print(x) #x must be nonzero here. Why?

print_nonzero([60, 0, 70, 0, 0, 80, 70])
```

```
60
70
80
70
```

241

Python "Modules"

Built-in: datetime, os, random ...

External: NLTK "package"

242

Example module: datetime

- https://docs.python.org/3/library/datetime.html
- import datetime
  - d = datetime.date(2016, 9, 10)
  - print(d.weekday())
  - print(d)
- Other ways of importing
  - from datetime import *
    #In this case, you'll say ...
    d = date(2016, 9, 10)

243

Another built-in module: random

- How to use it?
  - import random
  - x = random.random() #random float between 0 and 1
  - print(x)
  - n = random.choice(list) #returns one elem from list randomly
  - print(n)

245

List of useful Python modules

- Useful Python modules
  - https://wiki.python.org/moin/UsefulModules
- 20 Python libraries you can't live without
  - https://pythontips.com/2013/07/30/20-python-libraries-you-cant-live-without
- Come pre-packaged with Jupyter Notebook

253

Functions and scoping

258

Scoping rules

- Life of local variables (variables defined inside a function, including parameters)
  - Birth: initialization
  - Death: when function exits
- Scope: One function cannot access another function's local variables
  - Even if the other function is called
- How to share variables among functions?
  1. Parameters and return values
  2. Global variables (discouraged)

259

Error example

```
#This funciton is called by f
def g(x):
    x = 100 #Is x a local var of g()?

#f() is called from the shell
def f():
    a = 10 #a is a local var of f()
    g(a) #call g
    print(x) #Error: what is x?

#Call the function f
f()
```

Question: How can f() get the value of x from g()?

260