The Very First Course of Python

Changye Li

Institute of Health Informatics Student Group (IHI-ISG)
University of Minnesota

Jan. 2022

IHI STUDENT GROUP

What We Will Do Today I

1. Introduction
   - IHI-ISG
   - Programming Club
   - Your "Instructor" Today
2. Python Installment & Environment
   - Installment
3. Basic Python Syntax
   - Object references, or variables
   - Python Common Built-in Data Types
   - Calculator Programming Assignment
   - Advanced String Operations: Regular Expression
   - Other Useful Operations
   - Programming Assignments on String, List, Set and Dictionary
   - Data Type Conversion
4. Control Structures and Functions

IHI SG-UMN
IHI STUDENT GROUP

What We Will Do Today II

- Loops
  - `if` Loop
  - `while` Loop
  - `for` Loop

5. Custom & Better Functions
   - `def` Functions
   - `class` Functions
   - Improve Your Code
     - Python PEP 8
   - Doc-strings
   - Python Anonymous Function

6. Error Handling
   - Why Do We Need Error Handling?
   - Error Handling Examples

7. File Handling

IHI SG-UMN
IHI STUDENT GROUP

What We Will Do Today III

- Overview
- Opening Files in Python
- Opening Mode
- Writing to Files in Python
  - Writing Text Files
  - Writing CSV Files
  - Writing JSON Files

8. Collaboration
   - Pair Programming
   - Git & GitHub

IHI ISG-UMN
IHI STUDENT GROUP

What is IHI-ISG?

- We the official student group of Institute of Health Informatics (IHI)
- We serve as an IHI community liaison to other UMN Student Groups or Affiliates
- Our goal is to foster a community with a shared interest in the field of health informatics
- We encourage the academic and career development of students in the field of health informatics
- We offer networking and connecting students through information exchange on collaboration, workshops, and potential professional opportunities

IHI SG-UMN
IHI STUDENT GROUP

What is Programming Club?

- We are a subcommittee of ISG
- Our goal is to teach IHI students to programming on the introduction level
- We want IHI students to learn the ability of self-learning with programming with advanced topics
- Students will be able to program on the basic level, and able to learn more advanced topics (i.e., complex syntax, advanced application) on their own

ISG-UMN
IHI STUDENT GROUP

Who am I?

- Changye Li
- IHI Ph.D. student, Data Science and Informatics for Learning Health Systems track
- My research interest is Natural Language Processing (NLP) application on health and medical domain
- B.A. in Statistics, M.S. in Data Science

IHI SG-UMN
IHI STUDENT GROUP

Python Installment, Version, IDE

- Download Python *at least* **3.8** from Python official website
- **Always start with a virtual environment!** but not required for this workshop
- Chose your IDE, i.e., Visual studio code or PyCharm
  - Your python script extension is `.py`
- Jupyter notebook is also one of my favorite
  - Your jupyter notebook extension is `.ipynb`
- You can also try Python online compiler like this one
- We will use any IDE for the following workshop content

IHI ISG-UMN
IHI STUDENT GROUP

Your First Python Program

In your IDE, create a Python script named `hello_world.py`
In the Python script, type the following code:

```
print("Hello World")
```

What is the output of this script?

Variables

Python is an Object-oriented programming language, which means we need to store our data types as object – *object references* or *variables*. By doing that, Python allocates temporary memory to store the data we just declared

```
x = 55414 # int
y = "Hello World" # string
z = x
print(x, y, z)
```

What's the output of code above?
Does the computer "know" what 55414 is?

Rules of Naming Variables

- The start character should be a Unicode letter ("a", "b", ... "z", "A", "B", ..., "Z")
- The names of variables are case-sensitive. `var1`, `Var1`, `VAR1` are three different variables
- You can use underscore for your variables. For example `var_1`, but not for the first letter of your variable
- You cannot use Python keywords and predefined variables for your variables
- Make your variables easy to read and understandable. In general, don't name your variables like `var1`, `x`, `y`, `z`

IHI SG-UMN
IHI STUDENT GROUP

Short Quiz!

Which one is **NOT** a legal variable name in Python?

A `MyVal`

B `_my_val`

C `my-val`

D `myval1`

Python Keywords and Predefined Variables

and	continue	except	global	lambda	pass	while
as	def	False	if	None	raise	with
assert	del	finally	import	return	nonlocal	return
yield	break	elif	for	in	not	True
class	else	from	is	or	try	sum
super	tuple	type	zip			

Table: Python keywords and predefined variables

And many more ↓

```
print(dir(__builtins__))
```

IHI ISG-UMN IHI STUDENT GROUP

Python Common Built-in Data Types

- Integers (`int`): 1, 2, 3
- Booleans (`bool`): `True`, `False`
- Floats (`float`): 1.0, 2.0, 1.5
- Strings (`str`): "Hello World"
- Range (`range`): generates consecutive numbers with given range
- Tuples (`tuple()` or `(,)`): `(1.0, 2, True, "Hello World")`
- Lists (`list()` or `[]`): `[]`, `[1.0, 2, True, "Hello World"]`
- Sets (`set()`): `{1.0, 2, True, "Hello World"}`
- Dictionaries (`dict()`):
  `{1:1.0, "2": 2, "value": True, "message":"Hello World"}`

IHI SG-UMN
IHI STUDENT GROUP

Other Python Built-in Data Types Not Mentioned Here

- complex
- frozenset
- bytes
- bitarray
- memoryview

Immutable and Mutable Variables

Whenever an object is instantiated, it is assigned a unique object ID. If the object can change its state or content, then it is called mutable variable, otherwise it is immutable

- Immutable objects:
  `int`, `float`, `boolean`, `string`, `unicode`, `tuple`
- Mutable objects: `list`, `dict`, `set`

```
tuple_1 = (1, 2, 3, 4)
tuple_1[0] = 5
print(tuple_1)
message = "Hello World"
message[0] = "X"
print(message)
color = ["red", "blue", "gray"]
color[0] = "yellow"
print(color)
```

IHI SG-UMN
IHI STUDENT GROUP

Integers

Syntax	Description
`x+y`	adds number `x` and number `y`
`x-y`	subtracts `y` from `x`
`x*y`	multiplies `x` by `y`
`x/y`	divides `x` by `y`; always returns `float`
`x//y`	divides `x` by `y`, returns an `int` that truncates any fractional part of the result
`x%y`	returns the modules of `x/y`
`x**y`	x^y, `pow(x, y)`
`x==y`	returns `True` if two integers are the same
`abs(x)`	returns the absolute value of `x`

Table: Numeric operators and functions

Booleans

```
x = True
y = False
print(x and y)
print(x and True)
```

Booleans (Cont.)

Operation	Results
`x or y`	if x is false, then y, else x
`x and y`	if x is false, then x, else y
`not x`	if x is false, then true, else false
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Table: Python Boolean Operations and comparisons

IHI SG-UMN
IHI STUDENT GROUP

Truth Table: AND

Let's say T for True, F for False

x	y	x and y
F	F	F
F	T	F
T	F	F
T	T	T

Table: Truth Table: AND

Truth Table: OR

x	y	x or y
F	F	F
F	T	T
T	F	F
T	T	T

Table: Truth Table: OR

Truth Table: NOT

x	not X
F	T
T	F

Table: Truth Table: NOT

Floats

- Three types of floating-point value: `float`, `complex`, `decimal.decimal`
- `float` can be assigned with exponential notation: `print(8.9e-4)`

What's the difference between `1` and `1.0`?
Try `print(1 == 1.0)`

Short Programming Assignment

Write a very simple calculator in Python to implement the following features

- Addition
- Subtraction
- Multiplication
- Division

Open the `calculator.py` under `assignments` folder to complete this programming assignment

Strings

- String is a sequential text data

Strings can be created by single or double quotes, or simply `str()`

```
x = ""
y = str()
z = "Hello World"
```

But what if we want to create a string like this one with **one** string variable:

Hello
        World
I'm using Python

ISG-UMN
IHI STUDENT GROUP

Python Common String Escapes

Escape	Meaning
\\	backslash (\)
\'	single quote (')
\n	end-of-line terminator, Unix OS
\r	end-of-line terminator, old Mac OS
\r\n	end-of-line terminator, Windows and DOS OS
\b	backspace
\t	tab
\v	vertical tab

Table: Python Common String escapes

Programming quiz: use appropriate escape(s) to generate the above string with **one** variable

String Operations

Let's say `s = "Light ray"`. What if I want to extract individual characters?

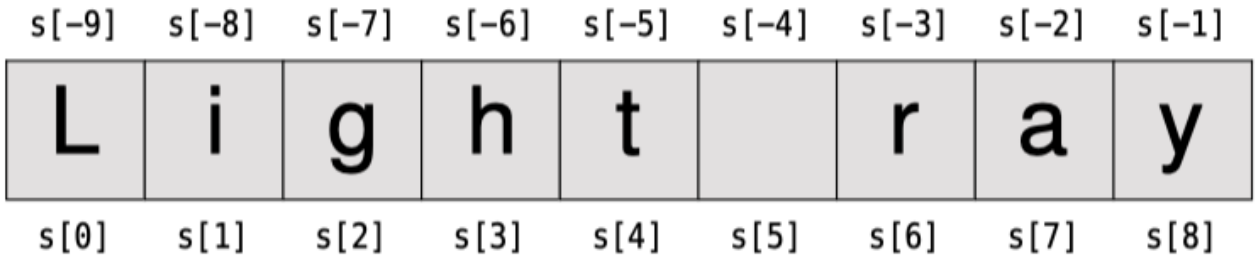

Figure: String index positions

- What if I want to extract `Light`? `s[:5]`
- What if I want to extract `ray`? `s[:-3]`
- What if I want to extract every two characters of `s` (`Lgtry`)? `s[::2]`

Short quiz: what if I want to extract `ght`?

IHI SG-UMN
IHI STUDENT GROUP

More on Strings

A sting object `s`

Syntax	Description
`len(s)`	returns the length of `s`
`s.capitalize()`	returns a copy of `s` with the first letter capitalized
`s.endswith(x, start, end)`	returns `True` if `s` (or during the `start:end` of `s`) ends with pattern `x`
`s.startswith(x, start, end)`	returns `True` if `s` (or during the `start:end` of `s`) starts with pattern `x`
`s.format()`	returns a copy of `s` formatted according to the given arguments
`s.lower()`	returns a copy of `s` that all characters are lowered

Table: More on String Operations

IHI SG-UMN
IHI STUDENT GROUP

More on Strings (Cont.): Format String

```
txt1 = "Hello Changye!"
print(txt1)
fname = input("Enter your first name:")
txt2 = "Hello {}!".format(fname.capitalize())
print(txt2)
```

Regular Expression

RE	Match
[wW]oodchucks	woodchucks or Woodchucks
[abc]	'a' or 'b' or 'c' characters from the string
[A-Z]	an upper case letter
[a-z]	a lower case letter
[0-9]	a single digit
[^A-Z]	NOT an upper case letter
[^a-z]	NOT a lower case letter
woodchucks?	woodchuck or woodchucks
colou?r	colour or color

Table: Regular Expression Examples

IHI ISG-UMN
IHI STUDENT GROUP

Regular Expression (Cont.)

RE	Match
beg.n	Any **single** character between beg and n
^	Start of the line
$	End of line
\b	Word boundary, $= [a-zA-Z0-9\_]$
\B	Non-word boundary
\d	Any digit, = [0-9]
\D	And non-digit, = [^0-9]
\s	Whitespace (space, tab)
\S	Non-whitespace

Table: Regular Expression Examples (Cont.)

IHI SG-UMN
IHI STUDENT GROUP

Regular Expression (Cont.)

RE	Match
*	Zero or more occurrences of the previous char or expression
+	One or more occurrences of the previous char or expression
?	Exactly zero or one occurrence of the previous char or expression
$\{n\}$	n occurrences of the previous char or expression
$\{n, m\}$	From n to m occurrences of the previous char or expression
$\{n,\}$	At least n occurrences of the previous char or expression
$\{, m\}$	Up to m occurrences of the previous char or expression

Table: Regular Expression Examples (Cont.)

IHI SG-UMN
IHI STUDENT GROUP

Regular Expression Exercise

- Write a RE to find cases of the English article *the* or *The*. Note, words like *theology* or *other* should not be selected by your RE
- Write a RE to check that a string contains only a certain set of characters (in this case a-z, A-Z and 0-9)
- Write a RE to match a string that has an 'a' followed by anything, ending in 'b'
- Write a RE to match a string that has an 'a' followed by two to three 'b'
- Write a RE to match a US phone number: 000-000-0000
- Write a RE to match an email address

You can validate your RE here

IHI SG-UMN
IHI STUDENT GROUP

Tuples

- Tuple is a sequential collection data type
- Tuple is ordered sequence data type

```
# tuple with more than 1 item
t = ("venus", -28, "green", "21", 19.74)
# tuple with 0 item
t = ()
# tuple with 1 item
t = ("venus",)
```

t[-5]	t[-4]	t[-3]	t[-2]	t[-1]
'venus'	-28	'green'	'21'	19.74
t[0]	t[1]	t[2]	t[3]	t[4]

Figure: Tuple index positions

IHI SG-UMN
IHI STUDENT GROUP

Lists

- List is a mutable, ordered sequence
- List uses the same index/sliding syntax as strings and tuples
- `l = []`
- `m = list()`

More on List

Syntax	Description
`l.append(x)`	appends item `x` to the end of list
`l.extend(m)`	appends all of iterable `m` 's items to the end of list `m`. Equivalent to `l += m`
`l[::-1]`	returns the reversed of `l`. Equivalent to `l.reverse()`
`l[1:]`	returns `l` with first item removed
`l[:-1]`	returns `l` with last item removed
`min(l)`	returns the minimal value of `l` if all items are digital numbers
`max(l)`	returns the max value of `l` if all items are digital numbers
`len(l)`	returns the length of `l`
`sum(l)`	returns the sum of all items for `l`, if all items are digital numbers

Table: List Operations

List Comprehension

I want to iterate a list to do some operations. What should I do? For example, for a list with all integers, I want to add 1 to every item.

```
l = [1, 2, 3, 4]
# the "old fashion" way
new_l = []
# use for loop to iterate list
for item in l:
    new_l.append(item+1)
print(new_l)
# the Python way
l = [item+1 for item in l]
print(l)
```

ISG-UMN
IHI STUDENT GROUP

List Comprehension Short Quizzes

What if I want to add 1 to **odd** items only?
What if I want to keep **even** items only from `l`?

```
[expression for item in iterable if condition]
[expression for item in iterable]
```

Sets

- Set is un-ordered collections that each item in set is hashable
- The value does not change during its lifetime, therefore both the items and their hash values are unique
- We cannot access items inside a set with indexes
- `s = {7, "veil", 0, -29, ("x", 11), 913}`
- `s = set()`

What will a set, `s = {7, "veil", 0, -29, ("x", 11), 913, "veil", 7}` look like?

Set Operations

Syntax	Description
`s.add(x)`	appends item `x` to the set
`s.clear()`	remove all items from the set
`s.difference(t)`	returns a new set that has every item that is in set `s` that is not in set `t`. Equivalent to `s-t`
`s.intersection(t)`	returns a new set that has each item that is in both set `s` and set `t`
`s.remove(x)`	remove item `x` from the set
`s.issubset(t)`	returns True if set `s` is equal to or a subset of set `t`. Equivalent to `s <= t`

Table: Set Operations

IHI STUDENT GROUP

Set comprehension also works, similar syntax to list comprehension

Dictionaries

- Dictionary is an un-ordered, mutable collection data types with key-value pairs.
- Keys are hashable (unique), but not for values
- `d = dict()`
- `d = {}`
- `d = {"name": "changye", "major": "health informatics", "id": 1234}`

IHI SG-UMN
IHI STUDENT GROUP

Dictionary Operations

Syntax	Description
`d.get(k)`	return the value associated with `k`
`d.items()`	returns the view of key-value pairs
`d.keys()`	returns the view of all keys of `d`
`d.values()`	returns the view of all values of `d`

Table: Dictionary Operations

The "view" is a read-only iterable object
Dictionary comprehension also works, similar syntax to list comprehension

IHI ISG-UMN
IHI STUDENT GROUP

Other Useful Operations

Operation	Results
<	strictly less than
<=	less than or equal
>	strictly greater
>=	greater than or equal
==	equal
!=	not equal
`is`	object identity
`is not`	negated object identity
`in`	returns `True` if a sequence with the specified value is present in the object
`not in`	returns `True` if a sequence with the specified value is not present in the object

Table: Other Useful Operations

ISG-UMN
IHI STUDENT GROUP

Short Programming Assignments

- Given a string `s = "Hello World"`, return a reversed version of it (`s = "dlroW olleH"`)
- Given a string `s = "Institute of Health Informatics"` and ignoring the cases, find the frequency of maximum occurring character in a python string. How about least frequent character?
- Given a string `s = "Institute of Health Informatics"`, return a list of unique characters, ignoring word orders and cases

IHI SG-UMN
IHI STUDENT GROUP

Data Type Conversion, Or Type Casting

- Implicit type conversion: Python automatically convert data types to others
- Explicit type conversion: we need to convert data types manually

Type Casting (Cont.)

```
x = 10
y = 10.1
print("x is type {}".format(type(x))
print("y is type {}".format(type(x))
x = x+y
print("x is type {}".format(type(x))
s = "10010"
c = int(s, 2)
print("{} is {} after converting to int base 2".format(s,
↪  c))
e = float(s)
print("{} is {} after converting to float".format(s, e))
c = tuple(s)
print("{} is {} after converting to tuple".format(s, c))
l = list(s)
print("{} is {} after converting to list".format(s, l))
```

How to Check Data Type

What do we need to know before casting data type conversions?

- Data type of the variable before conversion
- Desired data type

What if I don't know the data type before conversion?

```
x = 5
# suppose I forget it
# is it int?
if isinstance(x, int):
    print("yes it is an integer")
# or you can do this
print(type(x))
```

ISG-UMN
IHI STUDENT GROUP

if Loops

```
if boolean_expression_1:
    suite_1
# if you have more than one condition
elif boolean_expression_2:
    suite_2
# you can have zero or more elif causes
...
# else cause is optional
else:
    else_suite
# if you only have two conditions
expression_1 if boolean_expression else expression_2
```

Conditional Loop (Cont.)

Let's say we want to build a very simple reminder to greet users based on current weather. If it's a hot day, we would like to remind users to drink plenty of water. If it's a cold day, we would like to remind users to wear some warm clothes. Otherwise, we will greet users and state that it's a lovely day.

More specifically, hot day is defined as the temperature is greater than or equal to 85°F, cold day is defined as the temperature is less than or equal to 50°F, otherwise it's a lovely day.

How to write this reminder and greeting function in Python?

while Loop

We can execute a set of statements as long as a condition is true using while loop

```
while boolean_suite:
    while_suite
# else cause is optional
else:
    else_suite
```

for Loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string)

```
for expression in iterable:
    for_suite
# examples
fruits = ["apple", "peach", "pear"]
s = "Hello World"
for item in fruits:
    print(item)
for item in s:
    print(item)
# what if I want the value and its associated index?
for index, value in fruits:
    print(index, value)
```

Leave Loop: `break`, `continue`, `pass`

I've found what I looked for from a loop, then what should I do next?

- Continue to run your script
- "Stop" the loop using `break`, `continue`, `pass`
  - `break`: terminates the current loop and resume execution at the next statement
  - `continue`: "ignore" all remaining statements in the current iteration and move the control to the top of the loop
  - `pass`: something is required, but you don't have anything else to execute

IHI SG-UMN
IHI STUDENT GROUP

Leave Loop (Cont.)

```
for letter in "python":
    if letter == "h":
        break
    print("current letter is {}".format(letter))
print("======================")
for letter in "python":
    if letter == "h":
        continue
    print("current letter is {}".format(letter))
print("======================")
for letter in "python":
    if letter == "h":
        pass
    print("current letter is {}".format(letter))
```

SG-UMN
IHI STUDENT GROUP

Quizzes on Various Topics

- Given a list `l = list(range(11))`, find the **odd** numbers and save it to a new list, and print the new list
- What's the output of the following snippets? Don't copy and paste and run them on your laptop!

```
i = 1
while i<=10:
    if(i%2==0):
        print(i)
    i+=1
```

IHI SG-UMN
IHI STUDENT GROUP

Short Assignments

- Given an array of integers *nums* and an integer *target*, return indices of the two numbers such that they add up to *target*. For example, a list of `nums = [2,7,11,15], target = 9`, your script should return `nums = [0, 1]`. How about `nums = [3,2,4], target = 5`?
- Given an integer, return the integer that every single digit is reversed. For example, with input of 321, your script should return 123

IHI SG-UMN
IHI STUDENT GROUP

Write Your Own Functions

- A function is a block of code which only runs when it is called. You can pass data, known as parameters or arguments, into a function
- When you need to repeatedly use a piece of code more than 3 times, make it as function

```
def greet(fname):
    print("Hello {}!".format(fname))
greet("Changye")
# parameter can have default values
def greet(fname="Changye"):
    print("Hello {}!".format(fname))
greet()
```

Write Your Own Class

- `def` *do* things
- `class` *is* specific things, i.e., a special data structure with several functions
- If you have more than 2 functions can be applied to this special data structure, make it as `class`

Check `grade_book.py` for the `class` example under `assignments` folder

IHI SG-UMN
IHI STUDENT GROUP

Python PEP 8 - Python Enhancement Proposals: Write Your Code Easy to Read and Review

- Detailed plan here
- Install pylint under your environment for suggestions
- Use 4 spaces or 1 tab per indentation level, but **NOT** the mixture of spaces and tabs
- Limit all lines to a maximum of 79 characters.
- Avoid trailing whitespace anywhere

IHI SG-UMN
IHI STUDENT GROUP

Python PEP 8: Naming Styles

Type	Naming Convention	Examples
Function	Use lowercase word(s). Separate words by underscores to improve readability	`function`, `my_function`
Variable	Use lowercase single letter, word(s). Separate words with underscores	`v`, `var`, `my_variable`
Class	Start each word with a capital letter. **Don't** separate words with underscores	`Model`, `MyClass`
Method	Use lowercase word(s). Separate words with underscore	`class_method`, `method`
Constant	Use an uppercase single letter, word(s). Separate words with underscores	`CONSTANT`, `MY_CONSTANT`

Table: Python PEP 8 Naming Styles

IHI SG-UMN
IHI STUDENT GROUP

Comment Your Code, Write Documentation if Necessary

Take the reversed integer as an example

```
def reverse_int(input_int):
    # one-line docstring
    """Return the reversed integer with a given input
    ↪  integer."""
    # multiple lines of docstring
    """
    This is a reST style.

    :param input_int: the given integer to be reversed
    :returns: the reversed of given integer
    :raises keyError: raises an exception, we will talk
↪   about it very soon
    """
```

ISG-UMN
IHI STUDENT GROUP

Comment Your Code, Write Documentation if Necessary (Cont.)

```
def reverse_int(input_int):
    """
    This is an example of Google style.

    Args:
        input_int: description of this parameter

    Returns:
        This is a description of what is returned.

    Raises:
        KeyError: Raises an exception.
    """
```

Comment Your Code, Write Documentation if Necessary (Cont.)

```
def reverse_int(input_int):
    """
    This is sphinx style

    :param input_int: [ParamDescription], defaults to
↪   [DefaultParamVal]
    :type input_int: int

    :raises [ErrorType]: [ErrorDescription]

    :return: [ReturnDescription]
    :rtype: [ReturnType]
    """
```

More Programming Assignments

- Write a function to find if the input integer is palindrome. That's being said, if the input integer can be read the same backward as forward. For example, with input of 121, your function should return `True`, but with input of -121, it should return `False`

IHI SG-UMN
IHI STUDENT GROUP

Python Anonymous function

`lambda arguments: expression` is an anonymous function that is same as a regular python function but can be defined without a name

- reduce number of lines of codes compared to `def`
- function which is needed temporarily
- call it immediately at the end of definition

```
squares = lambda x: x*x
print("using lambda to get squares: {}".format(squares(2))
def squares(x):
"""
Ignore docstring for now
"""
    return x*x
print("using def to get squares: {}".format(squares(2))
```

ISG-UMN
IHI STUDENT GROUP

filter(), map(), reduce()

```
# given a list, return the even numbers from the list
from functools import reduce
from functools import filter
mylist = [2,3,4,5,6,7,8,9,10]
list_new  = list(filter(lambda x : (x%2==0), mylist))
print(list_new)
list_new  = list(map(lambda x : x%2, mylist))
print(list_new)
sum = reduce((lambda x,y: x+y), list1)
print(sum)
```

Quiz and Short Assignments

What is the output of the following code?

```
from functools import reduce
from functools import filter
words = ["bay", "cat", "boy", "fan"]
b_words = list(filter(lambda word: word.startswith("b"),
 ↪  words))
print(b_words)
words = ["mary", "had", "a", "little", "lamb"]
biggest_word = reduce(lambda x, y: x if len(x) > len(y)
 ↪  else y, words)
print(biggest_word)
```

Calculate the sum of `l=[1, 4 ,5, 2]`. `sum()` is not allowed

SG-UMN
IHI STUDENT GROUP

Why Do We Need Error Handling

- Prevents program from crashing if an error occurs: notify users of why the error occurred and gracefully exit the process
- Saves time debugging errors: the error messages are very useful information for the developer to track and debug
- Helps define requirements for the program: better definition of requirements and constraints

Try `print(6/0)`

IHI SG-UMN
IHI STUDENT GROUP

Error Handling (Cont.)

```
try:
    # test a block of code for errors
    try_suite
except exception_group1:
    # how to handle the error
    except_suite1
except exception_group2:
    # how to handle the error
    except_suite2
...
finally:
    # execute code, regardless of the result of try-catch
    ↪ blocks
    finally_suite
```

Error Handling Example

```
try:
    x = int(input("Please enter a number: "))
    y = 100 / x
except ValueError:
    print("Error: there was an error")
except ZeroDivisionError:
    print("Error: 0 is an invalid number")
except Exception:
    print("Error: another error occurred")
finally:
    print("Cleanup can go here")
```

A full list of build-in errors can be found here

Error Handling Example (Cont.)

```
lst = [4, 5, 0, 1]
def list_find(lst, target):
    try:
        index = lst.index(target)
    except IndexError:
        index = -1
    return index
list_find(lst, 2)
```

You can also raise an error and stop the program. Let's say we want to stop the program if the target is not in the list, using `raise IndexError("This number is not in the list")`. How should I change the function accordingly?

File Handling

- File is a named location on the system storage which records data for later access. It enables persistent storage in a non-volatile memory
- General flow
  - Open a file that returns a file handle
  - Use the handle to perform read or write action
  - close the file handle

Opening & Closing Files in Python

```
# open files in the current directory
f = open("test.txt")
# open files from another directory
f = open("full/path/to/file/test.txt")
f.close()
```

Opening Mode

Modes	Description
r	read-only, default
rb	read-only, in binary format
w	allows write-level access to the file. If the file already exists, the file will be overwritten, otherwise creates new file
wb	wirte-level access, in binary format
a	append mode, new content will be added at the end of file if file exists, otherwise creates new file
ab	append mode, in binary format

Table: Common File Opening Modes in Python

IHI STUDENT GROUP

Writing Text File

```
with open("test.txt", "w") as file_handle:
    file_handle.write("My first file.\n")
    file_handle.write("This is the file,\n")
    file_handle.write("which contains three lines.\n")
with open("test.txt", "r", encoding = "utf-8") as
↪   file_handle:
   content = file_handle.readlines()
for line in content:
    print(line)
```

IHI STUDENT GROUP

Writing CSV Files

Revisited the `grade_book.py` example, suppose we already have a dictionary with GPA and corresponding student record. How to write all student records to a single local file?

```
record_joe = {"student_id": 1, "name": "joe", "level":
↪  "bachelor", "gpa": 3.8}
record_jane = {"student_id": 2, "name": "jane", "level":
↪  "master", "gpa": 3.9}
record_justin = {"student_id": 3, "name": "justin",
↪  "level": "phd", "gpa": 3.9}
```

IHI SG-UMN
IHI STUDENT GROUP

Writing CSV Files (Cont.)

```
import csv # built-in module
with open("student_records.csv", "w") as csv_file:
    writer = csv.writer(csv_file)
    for key, value in record_joe.items():
        writer.writerow([key, value]
with open("student_records.csv", "a") as csv_file:
    writer = csv.writer(csv_file)
    for key, value in record_jane.items():
        writer.writerow([key, value]
# load file
with open("student_records.csv") as csv_file:
    reader = csv.reader(csv_file)
    record = dict(reader)
```

Writing CSV Files (Cont.)

```
import pandas as pd # external module, install it first
# merge dicts into a single dataframe
record_df = pd.DataFrame(columns = list(record_joe.keys()))
record_df = record_df.append(record_joe, ignore_index=True)
record_df = record_df.append(record_jane,
↪  ignore_index=True)
record_df = record_df.append(record_justin,
↪  ignore_index=True)
# write to local .csv file
record_df.to_csv("student_records.csv", index=False)
# load .csv file
record_df = pd.read_csv("student_records.csv")
```

Writing JSON Files

```
import json # built-in module
with open("student_records.json", "w") as out_file:
    json.jump(record_joe, out_file)
# load .json file
with open("sutdent_records.json") as json_file:
    record = json.load(json_file)
```

Pair Programming

Two programmers work together at one workstation. One, the *driver*, writes code while the other, the *observer* or *navigator* reviews each line of code as it is typed in. The two programmers switch roles frequently.

Pair programming in action: find your partner, and write a Python program to check if a string is palindrome

IHI STUDENT GROUP

Git & GitHub

- Git is a software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Guide on installing git can be found here
- GitHub is a provider of Internet hosting for software development and version control using Git

Generating SSH Key and Adding to GitHub

- SSH key pair is the "fingerprint" of your laptop
- SSH key pair can be generated by
  `ssh-keygen -t ed25519 -C "your_email@example.com"`
- Find your generated SSH key to using `cat ~/.ssh/id_rsa.pub`
- Copy your generated SSH key under Settings - SSH and GPG keys
- Common git commands can be found here
- A live demo

IHI STUDENT GROUP