Python Boot Camp Exercises

Module 1, 2024-25
HKU Business School

Contents

1. Basics 3
Exercise – greeting message 3
Exercise – temperature conversion 3
Exercise – even or odd 3
Exercise – voting age 3
Exercise – triangle type 3
2. Functions 4
Exercise – grade conversion 4
Exercise – factorial 4
Exercise – max of three 4
Exercise – leap year 4
3. Modules 5
Exercise – convert float to binary 5
Exercise – dart game 5
Exercise – roll two die 6
Exercise – rock paper scissors 6
4. Loops & iterations 6
Exercise – square root 6
Exercise – prime numbers 7
Exercise – repeated user inputs 7
Exercise – print a pattern 7
Exercise – Fibonacci Sequence 8
Exercise – guess a number 8
Exercise – sum of cubes 8
5. Strings 8
Exercise – extract a number 8
Exercise – print a hollow pyramid 9
Exercise – ticket discount 9
Exercise – swap first and last characters 10

Exercise – reverse a word 10
Exercise – sum of cubes (use strings) 10
Exercise – valid substrings 10
Exercise – rotate a letter 11
Exercise – print a letter pattern 12
6. Lists, Sets, & Dictionaries 12
Exercise – create a dictionary 12
Exercise – distinct three-digit numbers 13
Exercise – long words 13
Exercise – smallest number 13
Exercise – sort word by length 13
Exercise – verify ISBN-10 14
Exercise – distinct substring 14
Exercise – same frequency by one removal 15
Exercise – sort letters in a sequence 15
7. File I/O 15
Exercise – word frequency 15
Exercise – tennis court 15
Exercise – quiz scores 16

1. Basics

Exercise – greeting message

Given your first name and last name, print out a greeting message like follows.

```
>>> first = 'Chao'
>>> last = 'Ding'
>>> print(???)
Hello, Ding Chao.
```

Exercise – temperature conversion

Given a Celsius degree, convert it to Fahrenheit degree and print out a message like follows.
Note: °F = °C * 9/5 + 32.

```
>>> cel = 24.6
>>> fah = ???
>>> print(???)
24.6 Celsius degrees converted to Fahrenheit is 76.28
```

Exercise – even or odd

Given a positive integer, print out a message informing whether the integer is even or odd.

```
>>> num = 5
>>> ???
It is an odd number.
```

Exercise – voting age

Given a user's name and age, print a message informing whether the user is eligible to vote (18 years old and above) or not (below 18).

```
>>> name, age = 'Eric', 32
>>> ???
Eric, you can vote.
```

```
>>> name, age = 'Katherine', 14
>>> ???
Katherine, you cannot vote.
```

Exercise – triangle type

Given a triangle's lengths of three sides, print out the triangle's type:

- An equilateral triangle has three equal sides.
- A scalene triangle has three unequal sides.
- An isosceles triangle has at least two equal sides.

```
>>> x, y, z = 6, 6, 9
>>> ???
An isosceles triangle.
```

```
>>> x, y, z = 6, 7, 9
>>> ???
A scalene triangle.
```

```
>>> x, y, z = 9, 9, 9
>>> ???
An equilateral triangle.
```

2. Functions

Exercise – grade conversion

Build a function called grade(score) that takes a score (a positive integer) and returns a letter grade according to the following scoring table.

Score	Grade
>= 90	A
>= 80	B
>= 70	C
>= 60	D
< 60	F

```
>>> print('Your grade:', grade(96))
Your grade: A
```

Exercise – factorial

Build a function called fact(num) to calculate the factorial of a number (a non-negative integer).

```
>>> fact(3)
6
```

```
>>> fact(0)
1
```

Exercise – max of three

Build a function called max_of_three(a, b, c) to find the max of three numbers.

```
>>> max_of_three(6, -5, 1)
6
```

Exercise – leap year

Build a function called leap_year(year) to determine if year is a leap year or not. For simplicity, we consider a year to be valid if 0 < year < 9999.

Note: Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years if they are exactly divisible by 400. For example, the years 1700, 1800, and 1900 are not leap years, but the years 1600 and 2000 are.

```
>>> leap_year(2000)
'Leap year'
```

```
>>> leap_year('MSBA')
'WARNING: your input is not valid'
```

```
>>> leap_year(-1990)
'WARNING: your input is not valid'
```

```
>>> leap_year(1990.83)
'WARNING: your input is not valid'
```

```
>>> leap_year(1800)
'Regular year'
```

```
>>> leap_year(187000)
'WARNING: your input is not valid'
```

3. Modules

Exercise – convert float to binary

Write a program that creates a random floating number (between 0 and 1, rounded to 3 decimal places) and then converts the number to a binary number (0 or 1). The cutoff value is 0.5.

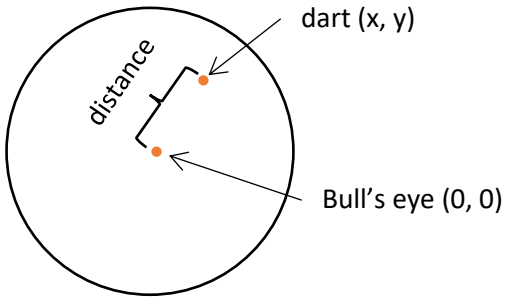
```
The original value is: 0.792
After conversion: 1
```

```
The original value is: 0.317
After conversion: 0
```

Exercise – dart game

Build a function called score(x, y) that returns a dart score, which is the distance between the dart location (x, y) and the bull's eye (0, 0) according to the following rule:

distance	score
<= 1	10
<= 5	5
<= 10	1
>10	"dart out of board"




```
>>> score(4, 3)
5
```

```
>>> score(-6, -3)
1
```

```
>>> score(5, -9)
'dart out of board.'
```

Exercise – roll two die

Write a program to simulate the game of "dice rolling" with two die. Randomly assign a number (between 1 and 6) to a dice, compare the values, and announce the winner.

```
Value of dice A is: 3
Value of dice B is: 6
B wins!
```

```
Value of dice A is: 4
Value of dice B is: 4
a draw!
```

Exercise – rock paper scissors

Write a program to simulate the game of "rock, paper, scissors" with two players. Randomly assign a choice to a player, compare the values, and announce the winner.

```
Player A chose scissors
Player B chose rock
Rock smashes scissors! Player B wins!
```

```
Player A chose rock
Player B chose rock
a draw!
```

4. Loops & iterations

Exercise – square root

Write a program that prints out all numbers that meet the following requirements:

1. The numbers must be between 1 and 100000.
2. The square root of (number + 100) is an integer.
3. The square root of (number + 200) is an integer.

Exercise – prime numbers

Build a function called `prime(stop)` that prints out all prime numbers between 0 and `stop`. Separate all prime numbers by a tab. `stop` must be an integer number greater than 0.

Note: prime numbers are whole numbers greater than 1, that have only two factors – 1 and the number itself. In other words, prime numbers are divisible only by the number 1 or itself.

```
>>> prime(30)
2	3	5	7	11	13	17	19	23	29
```

```
>>> prime(-3)
WARNING: Argument must be an integer number greater than 0.
```

Exercise – repeated user inputs

The `input` function takes a user input and returns a **string**. See an example below:

```
>>> input("Enter a number: ")
Enter a number: 5
'5'
```

Write a program which repeatedly asks a user to enter a number until the user enters “done”. Once “done” is entered, print out the total, count, and average of the numbers.

```
>>> Enter a number: 4
>>> Enter a number: 5
>>> Enter a number: 7
>>> Enter a number: done
-------------
Total: 16
Count: 3
Average: 5.3333
```

Exercise – print a pattern

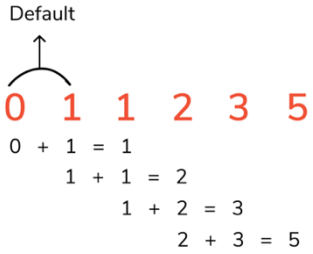
Write a program to construct the following pattern, using a nested `for` loop.

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

Exercise – Fibonacci Sequence

Write a program to get the Fibonacci Sequence between 0 to 50 considering that the first two numbers, 0 & 1, are given as default.

Note: The Fibonacci Sequence is the series of numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, ... Every next number is the addition of two numbers before it.



Exercise – guess a number

The input function takes a user input and returns a string. See an example below:

```
>>> input("Enter a number: ")
Enter a number: 5
'5'
```

Write a program to repeatedly ask a user to guess a number between 0 and 10 until the guess equals a target value. The target value is created at random.

```
>>> Take a guess: 5
Your guess is greater than the target.
>>> Take a guess: 3
Your guess is smaller than the target.
>>> Take a guess: 4
Bingo! You found it! It took you 3 guesses.
```

Exercise – sum of cubes

Write a program that searches all valid numbers (see rule below) between 100 and 1000. Print out all such numbers and separate them by a tab. A valid number can be presented as the sum of cubes of each digit, for example: $153 = 1^3 + 5^3 + 3^3$

```
The following numbers are valid:
XXX     XXX     …
```

5. Strings

Exercise – extract a number

Build a function called extract_number(text) that extracts, cleans up, and prints out the numbers in the following texts.

```
text1 = 'GDP per capita in 2019 is $24,564.'
text2 = 'GDP per capita in 2020 is $27,815.'
text3 = 'GDP per capita in 2021 is $30,179.'
text4 = 'GDP per capita in 2022 is $32,047.'
```

```
>>> extract_number(text1)
>>> extract_number(text2)
>>> extract_number(text3)
>>> extract_number(text4)
```

```
2019: 24564
2020: 27815
2021: 30179
2022: 32047
```

Exercise – print a hollow pyramid

Build a function called pyramid(n) that prints out a hollow pyramid of n rows. n is an integer greater than 3. On each row, the exact width of blanks between asterisks is decided by yourself, as long as the pattern looks like a pyramid. Only use one simple for loop; do NOT use nested for loop.

```
>>> pyramid(8)
       *
      * *
     *   *
    *     *
   *       *
  *         *
 *           *
*             *
```

Exercise – ticket discount

An airline company has a promotion campaign to provide the following special offers based on its customer's last name and the original ticket price. Note that offers can be accrued.

Offer	Condition on last name	Condition on original ticket price	Discount
1	1st character of last name is 'D' or 'd'		$10 discount
2	2nd character of last name is 'I' or 'i'	Original price >= $1000	$100 discount
3	3rd character of last name is 'R' or 'r'	Original price >= $5000	$500 discount
		$500 <= Original price < $5000	$300 discount

Build a function called discount(LaName, OriPrice) that takes the last name and the original ticket price, and returns the final price after applying eligible discounts. For simplicity, assume that arguments are always valid.

```
>>> discount('Wong', 10000) # no offer applies
10000
```

```
>>> discount('Ding', 6000) # offers 1 & 2 apply
5890
```

```
>>> discount('Li', 4560) # offer 2 applies
4460
```

```
>>> discount('DIREN', 5800) # offers 1 & 2 & 3 apply
5190
```

Exercise – swap first and last characters

Build a function called swap(word) that takes a string and swaps the first and the last characters.

```
>>> print(swap('abcd'))
>>> print(swap('12345'))
>>> print(swap('xy'))
>>> print(swap('k'))
dbca
52341
yx
k
```

Exercise – reverse a word

Build a function called RevWord(word) that takes a string and reverses the letters.

```
>>> RevWord("python")
'nohtyp'
```

Exercise – sum of cubes (use strings)

Write a program that searches all valid numbers (see rule below) between 100 and 1000. Print out all such numbers and separate them by a tab. A valid number can be presented as the sum of cubes of each digit, for example: $153 = 1^3 + 5^3 + 3^3$. Use strings only.

```
The following numbers are valid:
XXX    XXX    …
```

Exercise – valid substrings

Build a function called sub(MyStr) that takes a string and prints out "valid" substring(s). A substring is considered valid if its first letter is the same as the last letter. For example, the word 'presentation' has three valid substrings: 'ese', 'ntation', and 'tat'.

Requirement for MyStr:

- It must only include alphabetic letters. See examples 1 & 2 for some bad values.
- For simplicity, assume that you always pass a string when calling the function.
- It does not have to be an actual word/phrase.

Requirement for sub (MyStr):

- For invalid arguments, print out a warning message. See examples 1 & 2.
- A letter in upper case is considered the same as in lower case. See example 3.
- If the argument has valid substring(s), print out the total number of valid substring(s) and all such substring(s). Separate the valid substrings by some spaces. See examples 3, 4, & 5.
- If the argument has no valid substring, print out such result. See example 6.

```
>>> sub("HONG KONG")          # example 1
```

```
Please make sure the string includes only alphabetic letters.
```

```
>>> sub("20220314")          # example 2
Please make sure the string includes only alphabetic letters.
```

```
>>> sub("HONGkong")          # example 3
The string 'HONGkong' has 3 valid substring(s):
ONGko NGkon Gkong
```

```
>>> sub("presentation")      # example 4
The string 'presentation' has 3 valid substring(s):
ese   ntation    tat
```

```
>>> sub("program")           # example 5
The string 'program' has 1 valid substring(s):
rogr
```

```
>>> sub("spend")             # example 6
The string 'spend' has no valid substring.
```

Exercise – rotate a letter

Rotating a letter means to shift it through the alphabet, wrapping around to the beginning if necessary, so 'A' rotated by 3 is 'D' and 'Z' rotated by 1 is 'A'. Build a function called rotate_letter(letter, n) that accepts a letter and an integer n, and returns a new letter that is rotated by the amount of n.

```
>>> rotate_letter('t', 3)
'w'
```

```
>>> rotate_letter('Y', 5)
'D'
```

```
>>> rotate_letter('k', 20.1)
'WARNING: Your input is not valid.'
```

```
>>> rotate_letter('abc', 19)
'WARNING: Your input is not valid.'
```

```
>>> rotate_letter('D', -4)
'Z'
```

```
>>> rotate_letter('?', 8)
'WARNING: Your input is not valid.'
```

```
>>> rotate_letter(7, 109)
'WARNING: Your input is not valid.'
```

```
>>> rotate_letter(20, 'k')
```

```
'WARNING: Your input is not valid.'
```

Hint: `ord` is a built-in function which converts a character to a numeric value, and `chr` is a built-in function which converts a numeric value to a character. For example:

```
>>> ord('a')
97
```

```
>>> chr(65)
A
```

Exercise – print a letter pattern

Define your own function tri_letters(n) which accepts a positive integer n (1 < n < 10) and prints a triangle of letters like the example shown below. The number of rows is determined by n. On each row, letters are separated by one space. In your answer, test a few samples like below. Requirement: use a nested for loop.

```
>>> tri_letters(3)

A B C
 A B
  A
```

```
>>> tri_letters(20)

WARNING: your input is not valid
```

```
>>> tri_letters('abc')

WARNING: your input is not valid
```

```
>>> tri_letters(8)

A B C D E F G H
 A B C D E F G
  A B C D E F
   A B C D E
    A B C D
     A B C
      A B
       A
```

6. Lists, Sets, & Dictionaries

Exercise – create a dictionary

Use a loop to create a dictionary called new_dict where the keys are numbers between 1 and 9 (both included), and the values are the square of keys and converted to strings.

```
>>> new_dict
```

```
{1: '1', 2: '4', 3: '9', 4: '16', 5: '25', 6: '36', 7: '49', 8:
'64', 9: '81'}
```

Exercise – distinct three-digit numbers

Build a function called three_digit() that returns a list of three-digit numbers that meet the following rules:

1. Every number must be three digit long. 42 violates this rule.
2. Each digit must range between 1 and 4 (both numbers included). 156 violates this rule.
3. Each digit must be distinct. 141 violates this rule.
4. Numbers must be unique, i.e., no two numbers are the same.
5. Numbers in the resulting list are sorted in ascending order.

```
>>> print(len(three_digit()), three_digit()[4])
```

Exercise – long words

Build a function called long_words(text, k) that prints out a list of words that are longer than k characters from the text.

```
>>> text = 'The Python language is so much fun to learn'
>>> long_words(text, 4)
['Python', 'language', 'learn']
```

Exercise – smallest number

Build a function called find_smallest(seq) that returns the smallest number from a sequence (such as a list) of numbers.

```
>>> find_smallest([9, 41, 12, 3, 74, 15])
3
```

```
>>> find_smallest((1, 0, -43, 2.4))
-43
```

Exercise – sort word by length

Build a function called sort_word(text) that returns a list of words sorted by the length of each word in descending order from text.

```
>>> txt = "The python language is so much fun to learn."
>>> sort_word(txt)
[(8, 'language'),
 (6, 'python'),
 (6, 'learn.'),
 (4, 'much'),
 (3, 'fun'),
 (3, 'The'),
 (2, 'to'),
 (2, 'so'),
```

```
 (2, 'is')]
```

Exercise – verify ISBN-10

The International Standard Book Number (ISBN) is a numeric commercial book identifier. The ISBN-10 format has 10 digits (0 to 9) with the last digit being either a number or an X (e.g., 1-234-56789-X). Denote each digit by d_i, where $i = 1, \dots, 10$. The validity of ISBN-10 can be checked by the following:

$$\sum_{i=1}^{10} (11 - i) \cdot d_i \ \% \ 11 = 0$$

In the case the last digit is an X, then its value is '10'. For example, for an ISBN-10 of 3-598-21507-X:

$$10 \times 3 + 9 \times 5 + 8 \times 9 + 7 \times 8 + 6 \times 2 + 5 \times 1 + 4 \times 5 + 3 \times 0 + 2 \times 7 + 1 \times 10 = 264$$

Because 264 is divisible by 11 (i.e., 264 % 11 = 0), 3-598-21507-X is valid.

Build a function called `isbn_check(isbn)` that checks whether a given ISBN-10 is valid. Note that there are hyphens in ISBN-10 and they do not play a role.

```
>>> isbn_check('3-598-21507-X')
True
```

```
>>> isbn_check('3-598-41508-8')
False
```

Exercise – distinct substring

Build a function called `DistSub(text, k)` that finds the distinct substrings of length `k` in the `text`. Note that the distinct substrings should not be case-sensitive (see the last example). Print out all valid substrings in capital form, separated by a tab.

```
>>> DistSub('ABCAB', 8)
There is no such substring.
```

```
>>> DistSub('ABCAB', 5)
There is only one distinct substring in ABCAB, which is itself.
```

```
>>> DistSub('ABCAB', 3)
There are 3 distinct substrings in ABCAB. They are:
ABC   BCA   CAB
```

```
>>> DistSub('ABCAB', 2)
There are 3 distinct substrings in ABCAB. They are:
AB    BC    CA
```

```
>>> DistSub('ABCab', 2)
There are 3 distinct substrings in ABCAB. They are:
AB    BC    CA
```

Exercise – same frequency by one removal

Build a function call samefreq(text) that finds whether all other elements in the text share the same frequency after removing only one element.

```
>>> samefreq('CCCCDDDEEE') # remove letter C will do
Yes
```

```
>>> samefreq('xxxxxyzz')
No
```

```
>>> samefreq('abcdefgg') # remove letter g will do
Yes
```

Exercise – sort letters in a sequence

Define your own function sorting(seq) to sort any given sequence of letters in ascending order. Requirement: use a nested loop; do NOT use the built-in functions of min and max.

```
>>> sequence = ['b', 'z', 'a', 'd', 'k', 'o', 'e']
>>> sorting(sequence)
['a', 'b', 'd', 'e', 'k', 'o', 'z']
```

7. File I/O

Exercise – word frequency

Use the .txt file "twister.txt". Write a program to count the frequency of words in the file. Store your result in a dictionary called freq and print it out.

```
>>> print(freq)
{'how': 1, 'much': 3, 'wood': 4, 'would': 3, 'a': 4, 'woodchuck': 4,
 'chuck': 5, 'if': 2, 'could': 3, 'he': 2, 'as': 4, 'and': 1}
```

Finally, save your result in a json file called "freq.json" with an indentation of 3.

```
{
   "how": 1,
   "much": 3,
   "wood": 4,
   "would": 3,
```

Exercise – tennis court

There are 55 tennis courts information saved in the "tennis_raw.json" file. Extract each tennis court's name, district, number of courts, and phone number. Store your result in list called tennis, print out its length and the first 3 values.

```
>>> len(tennis)
55
```

```
>>> print(tennis[:3])
[['Ho Man Tin Sports Centre', 'Kowloon City', '2', '2762-7837'], ['J
unction Road Park', 'Kowloon City', '6', '2336-4638'], ['Ma Tau Wai
Service Reservoir Playground', 'Kowloon City', '4', '2713-7252/2711-
1532']]
```

Finally, save your result in a csv file named "tennis_clean.csv".

	A	B	C	D
1	Name	District	No_of_Courts	Phone
2	Ho Man Tin Sports Centre	Kowloon City	2	2762-7837
3	Junction Road Park	Kowloon City	6	2336-4638
4	Ma Tau Wai Service Reservoir Playground	Kowloon City	4	2713-7252/2711-1532
5	Tin Kwong Road Tennis Court	Kowloon City	4	2711-1532
6	North District Sports Ground	North	6	2679-4913

Exercise – quiz scores

Use the csv file "student_response.csv". This data file is a record of 55 students' answers to a quiz of 12 multiple choice questions. For example, student S1's answer is ACBCCDABDABD.

student_name	answer
S1	ACBCCDABDABD
S2	ACDCDCDAADCD

The correct solution is **ADBCCBABCCAD** with each question valued at 1 point. Therefore, student S1 earns 7 points (the ones with an underscore are correct: A~~C~~BCC~~D~~AB~~DAB~~D). Note that several students' answers have a different format that needs to be cleaned up:

S6	A B B C C B B C C C A D
S15	1.D 2.D 3.B 4.B 5.C 6.A 7.B 8.C 9.C 10.C 11.A 12.D
S26	B:C:B:C:C:D:B:C:B:B:A:D
S34	A-A-D-B-C-D-C-C-B-C-C-C
S44	1-A 2-B 3-B 4-C 5-B 6-D 7-C 8-C 9-C 10-D 11-C 12-C

Your tasks are to **clean up** (task 1) the answers when necessary, **calculate** (task 2) all 55 students' scores and **store** the result in an appropriate data structure called final_result, and **write** (task 3) the result to a csv file named "score.csv". It should have the following header: student_name, clean_answer, score. See a snapshot below:

student_name	clean_answer	score
S1	ACBCCDABDABD	7
S2	ACDCDCDAADCD	3
S3	ADDCCBBDCCBD	8
S4	ABCCCBCBACAD	8
S5	ACCBBCACBBAC	3
S6	ABBCCBBCCCAD	9
S7	ACCCDDBAADAD	4

In your notebook, **print out** (task 4) all the "odd-numbered" (i.e., S1, S3, S5, etc.) students' names, clean answers, and scores. There is no requirement on the printout format.