Practical Python for High-school Students

Phong Le (https://dple.github.io)

About this textbook

This textbook is prepared for high school students who want to learn to code and solve problems with Python. You may find many books or online resources that teach Python to beginners. As I could not find any single source that balances coding and problem-solving at the high school level, I decided to 'collect' from many sources (some of them are mentioned in the reference section below) and gather them in this textbook. Of course, the balance I mentioned here is just my personal view, and you may not need to agree with that. If you, by any chance, are reading this book, I hope you enjoy and find it helpful for your study.

References

Python Tutorial
PYnative
HackerRank
Learn Python
Learn Python Programming
Canadian Computing Competition - Computing Contests - University of Waterloo

Table of Contents

About this textbook 1
References 1
Chapter 1: Introduction to Python 5
1. Your First Python Code 5
1.1. Input 5
1.2. Output 5
1.3. Comment 7
1.4. Indentation 8
1.5. Assignment Statements 8
Exercises 9
2. Variables and Expressions 11
2.1. Variable Declaration 11
2.2. Data Types 12
2.3. Variable Names 14
2.4. Multi Words Variable Names 15
2.5. Assign Multiple Values 16
2.6. Print variables 17
2.7. Expressions 17
Exercises 18
Chapter 2. Basics 20
1. Operations 20
1.1. Arithmetic Operators 20
1.2. Assignment operators 20
1.3. Comparison operators 23
1.4. Logical operators 23
1.5. Identity operators 24
1.6. Membership operators 24
1.7. Bitwise operators 24
Exercises 26
2. If-else Statements 29
Exercises 29
3. Loops 33

3.1. For loop 33
3.2. While loop 34
Excercises 34
5. Function 38
5.1. Function with arguments 39
5.2. Function With Arbitrary Arguments 39
5.3. Recursive Function 40
Exercises 41
Chapter 3: Math 43
1. Built-in math functions 43
2. Math Package 44
2.2. Functions 44
2.2. Constants 44
3. Random Package 44
Exercises 45
Chapter 4: Mid-term Projects 49
Project 1: Rock, Paper, Scissors 49
Project 2: Number Guessing 49
Project 3 Adventure game 50
Project 4 Rolling Dice 51
Chapter 5: Data Structures 52
1. String 52
1.1. Declaration 52
1.2. Basic Operations 52
1.3. Methods 53
Exercises 58
2. List 64
2.1. Declaration 64
2.2. Methods 65
Exercises 65
3. Sets 69
3.1. Create a Set in Python 69
3.2. Methods for Set in Python 69

Exercises 71
4. Dictionary 77
4.1. Methods for Dictionary in Python..77
Exercises 78

Chapter 1: Introduction to Python

Python is a high-level, interpreted programming language that is widely used. It is usually considered as one of the programming languages to learn for beginners because of its simplicity. Python's syntax is designed to be simple, readable and concise compared to compiled programming languages such as C++, Java. This makes Python easy and even fun to learn first.

With its extensive library of modules and packages, Python is popularly used not only for secondary schools' students with school problems, programing games with PyGame, but also allows web development, scientific computing, data science, artificial intelligence, and more. Additionally, Python is highly customizable, allowing users to create their own modules and packages to extend the language's capabilities.

1. Your First Python Code

1.1. Input

Python has a built-in function, `input()` for getting input from the user. This function reads a line of text from the user and returns it as a string. For example:

```
name = input("Enter your name: ")
print("Hello, " + name + "!")
age = int(input("Enter an integer: "))
print("You're " + age + "years old!")
```

Remark: the function `int`() converts a numeric string into an integer.

1.2. Output

To output information to the user, Python has several functions such as `print()`, `format()`, `f-strings`, etc. The most commonly used is the `print`() function, which outputs text or variables to the screen. For example, the following code prints the message "Hello, world!" to the console:

```
print("Hello, world!")
```

The full parameters of the function print as follow:

```
print(object(s), sep=separator, end=end, file=file,
flush=flush)
```

Parameter	**Description**
object(s)	Any object, and as many as you like. Will be converted to string before printed
sep='separator'	Optional. Specify how to separate objects, if there is more than one. Default is ' '
end='end'	Optional. Specify what to print at the end. Default is '\n' (line feed)
File	Optional. An object with a write method. Default is sys.stdout
Flush	Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False

Output Formatting
There are several ways to format an output in Python. The `format`() method can be used to insert variables into a string. For example:

```
print('Hello, {}!'.format('world'))
print('{1} is hot, {0} is cold!'.format('Winter', 'Summer'))
age = 25
print('You're {} years old!'.format(age))
```

Remark: In Python, a string can be put in a pair of single or double quotes (i.e., ' ' or " ").

Alternatively, Python 3.6 introduced f-strings, which provide a more concise way to format output using variables. For example:

```
age = 25
print(f"You're {age} years old!")
```

In addition to console output, Python can also read from and write to files using built-in functions such as `open()`, `read()`, `write()`, and `close()`. These functions allow Python to interact with data stored in external files, such as text files, CSV files, and JSON files.

Escape Characters

In Python strings, **the backslash "\"** is a **special character**, also called the "**escape**" character. It is used in representing certain whitespace characters: "\t" is a tab, "\n" is a newline, and "\r" is a carriage return. Escape characters can be classified as non-printable characters when backslash precedes them, that is, the print statements do not print escape characters.

Code	Description
\’	Single quotation

\\\\	Backslash
	
	`print(”\\ is a backslash”)`
	
	`Output`
	`\ is a backslash`

\n	New Line
	`print(”apple\norange”)`
	
	`Output`
	`apple`
	`orange`

\r	Carriage return

\t	Insert Tab between two texts
	
	`print(”apple\torange”)`
	
	`Output`
	`apple orange`

\b	Backspace
\f	Form feed

1.3. Comment

Comments start with a #, and Python will render the rest of the line as a comment:

```
# This is a comment.
print("Hello, World!")
```

Multiline comments: Comment more than 1 lines:

```
"""
This is
an awesome
comment.
"""
print("Hello, World!")
```

1.4. Indentation

Indentation refers to the spaces at the beginning of a code line. In Python, indentation is used to indicate a block of code. For example, a correct code for a function is as below:

```
def print_name(name):
    print("Hello " + name)
    return
```

The following code will get error:

```
def print_name(name):
print("Hello " + name)
return
```

The number of spaces can be varied, the most common use is four, but it has to be at least one. However, You have to use the same number of spaces in the same block of code, otherwise Python will give you an error as in the below example:

```
def print_name(name):
    print("Hello " + name)
  return
```

1.5. Assignment Statements

Assignment operators in Python assign the right-hand side values to the operand that is present on the left-hand side. The basic assignment operator is “=”. For example, x = 5 assigns 5 to the variable x.

Exercises

Exercise 1: **Hello World**

Print “Hello World” to the screen.

Exercise 2: Greeting
Get a name from input, then print “Hello” + name to the screen.

Sample Input:
Show an asking question: What is your name?
Bo

Output:
Hello Bo!

Exercise 3:
Print four strings ‘The’, ‘life’, ‘is’, ‘beautiful’ to the screen in format ‘The-life-is-beautiful’.

Exercise 4:
Get day, month and year, then print them out in the format YYYY-MM-DD.

Exercise 5:
Print the statement “Python is an awesome programming language to learn for beginners” to the screen. Each word will be printed per line.

Exercise 6: Get two numbers from STDIN, then print the sum to the screen

Sample Input
Enter the number 1: 2
Enter the number 2: 3

Sample Output
Sum of two numbers is 5

Exercise 7: Get three numbers in one line from STDIN, separated by a space, then print their product.

Sample Input
Enter three numbers: 1: 2 5 7

Sample Output
The product of three numbers is 70

Exercise 8: String Format
Get name and age from STDIN, then print to the screen using string.format()

2. Variables and Expressions

A variable is a name that represents a value or an object in a program. You can think of a variable as a container for storing data that you can use later in the program. As other programming languages, a variable in Python is a value that can change, depending on conditions or on information passed to the program.

2.1. Variable Declaration

In Python, variables are created when you assign a value to it. If you want to store a number in a variable called “my_number”. You can do this by assigning a value to the variable using the assignment operator "=". For example:

```
my_number = 5
```

You can also change the value of a variable by assigning a new value to it. For example:

```
my_number = 7
```

Now, the variable “my_number” holds the value 7 instead of 5.

In Python, variables can hold different types of values, such as numbers, strings, lists, and more. For example:

```
my_string = "Hello, World!" # assign a string to a variable "my_string"
my_list = [4, 2, 5, 6, 9]"  # variable "my_list" stores a list of int
```

In Python, variables do not need to be declared with any particular type, and can even change type after they have been set.

```
x = 4       # x is of type integer (int)
x = "Sally" # x is now of type string (str)
print(x)
```

Case-sensitive: Variable names are case-sensitive. This will create two variables:

```
a = 4
A = "Sally"
# A will not overwrite a
```

2.2. Data Types

Variables can store data of different types. Python has the following built-in data types:

Data Type	**Description**

String: `str`	A string is a sequence of characters, enclosed in either single, double, or triple quotes. The triple quotes (“””) can be used for writing multi-line strings. Strings are used to represent text data in a program.
	
	`x = 'A'`
	`y = "B"`
	`z = """`
	`  C`
	`  """`
	
	`print(x)  # prints A`
	`print(y)  # prints B`
	`print(z)  # prints C`
	
	`print(x + y)  # prints AB - concatenation`
	
	`print(x*2) # prints AA - repetition operator`

Numeric types: `int`, `float`, `complex`	Include integer (int), number with a decimal point (float) and complex numbers (complex)
	
	`x = 2           # int`
	`x = int(2)       # int`
	
	`x = 2.5          # float`
	`x = float(2.5)   # float`
	
	`x = 100+3j       # complex`
	`x = complex(100+3j)   # complex`

Sequence types: `list`, `tuple`, `range`	Include list, tuple and range
Map type: `dict`	Dictionary is an ordered set of a key-value pair of items. A key can hold any primitive data type whereas value is an arbitrary Python object. The entries in the dictionary are separated with the comma and enclosed in the curly braces {, }. charsMap = {1:'a', 2:'b', 3:'c', 4:'d'}

Set types: `set`, `frozenset`	The set in python can be defined as the unordered collection of various items enclosed within the curly braces {, }.
	The elements of the set can not be duplicated. The elements of the python set must be immutable.
	Unlike sequence types such as list, tuple, range there is no index for set elements. It means we can only loop through the elements of the set.
	
	`digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}`
	`print("looping through the set elements ... ")`
	`for i in digits:`
	`    print(i)     # prints 0 1 2 3 4 5 6 7 8 9 in new lines`

Boolean type: `bool`	bool values are the two constant objects False and True. They are used to represent truth values. In numeric contexts, they behave like the integers 0 and 1, respectively.

Binary types: `bytes`, `bytearray`, `memoryview`	bytes and bytearray are used for manipulating binary data. The memoryview uses the buffer protocol to access the memory of other binary objects without needing to make a copy.
	
	bytearray objects are always created by calling the constructor bytearray(). These are **mutable** objects.
	
	`x = b'char_data'`
	`x = b"char_data"`
	
	`y = bytearray(5)`
	
	`z = memoryview(bytes(5))`

	`print(x)   # b'char_data'`
	`print(y)   # bytearray(b'\x00\x00\x00\x00\x00')`
	`print(z)   # <memory at 0x014CE328>`

NoneType	None

Casting: If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

Get the type: You can get the data type of a variable with the type() function.

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

2.3. Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords such as if, while, and, or, …

Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Illegal variable names:

```
2myvar = "John"
my-var = "John"
my var = "John"
```

2.4. Multi Words Variable Names

Variable names with more than one word can be difficult to read. There are several techniques you can use to make them more readable:

Camel Case: Each word, except the first, starts with a capital letter:

```
myVariableName = "John"
```

Pascal Case: Each word starts with a capital letter:

```
myVariableName = "John"
```

Snake Case: Each word is separated by an underscore character:

```
my_variable_name = "John"
```

2.5. Assign Multiple Values

Many Values to Multiple Variables: Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

One Value to Multiple Variables: And you can assign the *same* value to multiple variables in one line:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Unpack a Collection: If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*. For example, unpacking a list:

```
fruits = ["Apple", "Banana", "Cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

Output:

```
Apple
Banana
Cherry
```

2.6. Print variables

Once a variable was created and assigned a value, it can be printed. The Python print() function is often used to output variables.

```
x = "Python is awesome"
print(x)
```

In the print() function, you output multiple variables, separated by a comma:

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

You can also use the + operator to output multiple variables:

```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

For numbers, the + character works as a mathematical operator:

```
x = 5
y = 10
print(x + y)
```

2.7. Expressions

A sequence of operands and operators, like x + y - 5, is called an expression. Python supports many operators for combining data objects into expressions.

Exercises

Exercise 1: Create a string variable named fruitname, assign the value Apple to it, then print it out to the screen.

Exercise 2: Create an integer variable named n and assign the value 50 to it, then print it out to the screen.

Exercise 3: Write a program that reads in a whole number n, and prints n + 1 out to the screen.

Exercise 4: Interaction
Write a program that asks the user a series of questions about his **name**, **eye_color**, **hair_color**, **height** and **tall**. Then print out all information in a sentence.

Sample Input

What is your name?:	Bo
What is the color of your eye?:	black
What is the color of your hair?:	red
What is your height?:	140
What is your weight?:	80

Sample Output
Bo has black eyes and red hair. He is 140 cm and weighs 80 pounds..

Exercise 5: Integer to String
Write a program to assign an integer to a variable, then convert it to a string, then print the string to the screen.

Exercise 6: String to Integer
Write a program to assign two numeric strings to two variables, convert them to integers, then print the product to the screen.

Exercise 7:
Write a program that asks the user to input marks for 4 courses, and then displays the average mark.

Exercise 8: **Circle**

Write a program that will ask the user for the **radius** of a circle. The program will then calculate the **circumference** and **area** of the circle and output the results clearly. Use a variable for words marked in **bold**. **Hint**: Use pi = 3.14

Exercise 9: **Rectangle**
Write a program that will ask the user for the **length** and **width** of a rectangle. The program will then calculate the **perimeter** and **area** and output the results clearly. Use variables for the words marked in **bold**.

Chapter 2. Basics

1. Operations

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

1.1. Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	**Name**	**Example**
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

1.2. Assignment operators

Assignment operators are used to assign values to variables:

Operator	Name	Description
=	Assignment Operator	It assigns the right-hand side expression value to the operand present on the left-hand side.
+=	Addition Assignment Operator	This operator adds left and right operands, and after that, it assigns the calculated value to the left-hand operand.
-=	Subtraction Assignment Operator	This operator subtracts the right operand from the left operand and assigns the result to the left operand.
*=	Multiplication Assignment Operator	This operator will multiply the left-hand side operand by the right-hand side operand and, after that, assign the result to the left-hand operand.
/=	Division Assignment Operator	This operator will divide the left-hand side operand by the right-hand side operand and, after that, assign the result to the left-hand operand.
%=	Modulus Assignment Operator	This operator will divide the left-hand side operand by the right-hand side operand and, after that, assign the reminder to the left-hand operand.
**=	Exponentiation Assignment Operator	This operator raises the left-side operand to the power of the right-side operand and assigns the result to the left-side value.
//=	Floor Division Assignment Operator	This operator will divide the left operand by the right operand and assign the quotient value (which would be in the form of an integer) to the left operand.
&=	Bitwise AND Assignment Operator	This operator will perform the bitwise operation on both the left and right operands and, after that, it will assign the resultant value to the left operand.
|=	Bitwise OR Assignment Operator	This operator will perform the bitwise or operation on both the left and right operands and, after that, it will assign the resultant value to the left operand.
^=	Bitwise XOR Assignment Operator	This operator will perform the bitwise XOR operation on the left and right operands and, after

		that, it will assign the resultant value to the left operand.
>>=	Bitwise Right Shift Assignment Operator	This operator will right-shift the left operand by the specified position, i.e., b, and after that, it will assign the resultant value to the left operand.
<<=	Bitwise Left Shift Assignment Operator	This operator will shift the left operand by the specified position, i.e., b, and after that, it will assign the resultant value to the left operand.

Example:

Operator	**Example**	**Same As**
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

1.3. Comparison operators

Comparison operators are used to compare two values:

Operator	**Name**	**Example**
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

1.4. Logical operators

Logical operators are used to combine conditional statements:

Operator	**Description**	**Example**
`and`	Returns True if both statements are true	`x < 5 and  x < 10`
`or`	Returns True if one of the statements is true	`x < 5 or x < 4`
`not`	Reverse the result, returns False if the result is true	`not(x<5 and x<10)`

X	**Y**	**not X**	**X and Y**	**X or Y**
True	True	False	True	True
True	False	False	False	True
False	True	True	False	True
False	False	True	False	False

1.5. Identity operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	**Description**	**Example**
`is`	Returns True if both variables are the same object	`x is y`
`is not`	Returns True if both variables are not the same object	`x is not y`

1.6. Membership operators

Membership operators are used to test if a sequence is presented in an object:

Operator	**Description**	**Example**
`in`	Returns True if a sequence with the specified value is present in the object	`x in y`
`not in`	Returns True if a sequence with the specified value is not present in the object	`x not in y`

1.7. Bitwise operators

Bitwise operators are used to compare (binary) numbers. Bitwise operators treat operands as sequences of binary digits and operate on them bit by bit. The following operators are supported:

Operator	**Example**	**Meaning**	**Result**
&	a & b	bitwise AND	Each bit position in the result is the logical AND of the bits in the corresponding position of the operands. (1 if both are 1, otherwise 0.)

|	a | b	bitwise OR	Each bit position in the result is the logical OR of the bits in the corresponding position of the operands. (1 if either is 1, otherwise 0.)
~	~a	bitwise negation	Each bit position in the result is the logical negation of the bit in the corresponding position of the operand. (1 if 0, 0 if 1.)
^	a ^ b	bitwise XOR (exclusive OR)	Each bit position in the result is the logical XOR of the bits in the corresponding position of the operands. (1 if the bits in the operands are different, 0 if they are the same.)
>>	a >> n	Shift right n places	Each bit is shifted right n places.
<<	a << n	Shift left n places	Each bit is shifted left n places.

Here are some examples:

```
>>> '0b{:04b}'.format(0b1100 & 0b1010)
'0b1000'
>>> '0b{:04b}'.format(0b1100 | 0b1010)
'0b1110'
>>> '0b{:04b}'.format(0b1100 ^ 0b1010)
'0b0110'
>>> '0b{:04b}'.format(0b1100 >> 2)
'0b0011'
>>> '0b{:04b}'.format(0b0011 << 2)
'0b1100'
```

Exercises

Exercise 1: **Basic Operations**

Read two integers n & m from STDIN, then print 5 lines :

1. The first line contains the sum of the two numbers.
2. The second line contains the difference of the two numbers (first - second).
3. The third line contains the product of the two numbers.
4. The fourth line contains the remainder when dividing n by m.
5. The fifth line contains the exponent of n to m, i.e., n^m

For example, if the user entered the numbers 2 and 3, you would output the results of 2+3, 2-3, 2*3, 2/3 and 2^3 (2 to the exponent 3).

Exercise 2: **Greetings**
Ask the user their name, age, and the current year. Greet the user and tell them the year they were born.

Exercise 3: **Division**
Get two integers, a and b, from STDIN. Add logic to print two lines. The first line should contain the result of integer division, a//b. The second line should contain the result of float division, a/b. No rounding or formatting is necessary.

Exercise 4: **Sum and Average**
Write a program that asks the user for three numbers and then outputs the sum (total) and the average of the three numbers.

Exercise 5: **Fahrenheit to Celsius**
Write a program that reads a temperature in degrees Fahrenheit and prints out the corresponding temperature in degrees Celsius (C = (F - 32) * 5 / 9).

Exercise 6: Celsius to Fahrenheit

Write a program that reads a temperature in degrees Celsius and prints out the corresponding temperature in degrees Fahrenheit (F = (C * 9 / 5) + 32).

Exercise 7: **Cupcake Party**
Problem Description
A regular box of cupcakes holds 8 cupcakes, while a small box holds 3 cupcakes. There are 28 students in a class and a total of at least 28 cupcakes. Your job is to determine how many cupcakes will be left over if each student gets one cupcake.

Input Specification
The input consists of two lines.
- The first line contains an integer R ≥ 0, representing the number of regular boxes.
- The second line contains an integer S ≥ 0, representing the number of small boxes.

Output Specification
Output the number of cupcakes that are left over.

Sample Input
2
5
Output for Sample Input
3
Explanation of Output for Sample Input
The total number of cupcakes is 2 × 8 + 5 × 3 which equals 31. Since there are 28 students, there are 3 cupcakes left over.

Exercise 8:
Write a program that asks the user to type the price without tax of one kilogram of apple, the weight (in kilograms) he bought and the tax in percent units. The program will output the money including taxes he must pay.

Exercise 9: Distance of two points
Write a program that asks the user to type the 2D coordinate of 2 points, A and B (in a plane), and then outputs the distance between A and B.
Hint: If A = (a, b) and B = (c, d), then the distance between A & B is sqrt((a - c)^2 + (b - d)^2).

Sample Input
Enter the coordinate of A: 2 4
Enter the coordinate of B: 5 8

Sample Output
Distance between A and B is: 5

Exercise 10: Inner product

Write a program that asks the user two vectors V & W of dimension n, then calculate their inner product.
Hint: If V = (x_1, x_2, …, x_n) and W = (y_1, y_2, …, y_n), then their inner product is calculated as

<V, W> = x_1, y_1 + x_2, y_2 + … + x_n, y_n

Sample Input

```
Enter the dimension of vectors: 3
Enter vector V: 2 4 1
Enter vector W: 5 -3 -1
```

Sample Output

```
Inner product <V, W> is: -3
```

2. If-else Statements

if the condition(s) is/are satisfied, then commands 1 will be executed, otherwise, commands 2 will.

-

```
if <condition(s)>:
      <commands 1>
else:
      <commands 2>
```

Exercises

Exercise 1: **Income Tax**
Write a program that reads in a person's annual income and then computes and outputs their income tax liability assuming the following:

- there is a $15,000 tax free allowance
- the first $30,000 of taxable income is taxed at the 'lower rate' of 20%
- the next $30,000 of taxable income is taxed at the 'basic rate' of 30%
- The next $35,000 of taxable income is taxed at the ‘higher rate’ of 40%
- any further income is taxed at the 'highest rate' of 50%

Exercise 2: Ask and response
Write a program that asks the user

How many days per week are there?

Check the user’s response then print “Correct” if the response is 7; print “Almost correct” if the response is 6 or 8. Otherwise, print “Incorrect”

Exercise 3: **Leap Year**
Write a program that reads from STDIN a year (0< year < 2024), then determines if the year is a *leap year* or not. Print *True* if leap year, otherwise, print *False*.

Hint: a leap year is a year that is divisible by 4, and not exactly divisible by 100, unless it is divisible by 400.

Exercise 4: **Days of a month**
Given a *month* (1 <= month <= 12), print the number of days in that *month*. For example: January has 31 days, February has 28 days in a normal year and 29 days in leap year, etc. If the user does not enter an integer in the range 1 to 12 then output an error message: "A month must be between 1 and 12".

Exercise 5: **Deliv-e-droid Problem**

In the game, Deliv-e-droid, a robot droid has to deliver packages while avoiding obstacles. At the end of the game, the final score is calculated based on the following point system:

- Gain 50 points for every package delivered.
- Lose 10 points for every collision with an obstacle.
- Earn a bonus 500 points if the number of packages delivered is greater than the number of collisions with obstacles.

Your job is to determine the final score at the end of a game.

Input Specification
The input will consist of two lines. The first line will contain a non-negative integer P, representing the number of packages delivered. The second line will contain a non-negative integer C, representing the number of collisions with obstacles.

Output Specification The output will consist of a single integer F, representing the final score.

Sample Input 1
5
2

Output for Sample Input 1
730

Explanation of Output for Sample Input 1
There are 5 packages delivered, so 5 × 50 = 250 points are gained. There are 2 collisions, so 2 × 10 = 20 points are lost. Since 5 > 2, a bonus 500 points are earned. Therefore, the final score is 250 − 20 + 500 = 730.

Sample Input 2
0
10

Output for Sample Input 2
-100

Explanation of Output for Sample Input 2
There are 0 packages delivered, so 0 × 50 = 0 points are gained. There are 10 collisions, so 10 × 10 = 100 points are lost. Since 0 ≤ 10, no bonus points are earned. Therefore, the final score is 0 − 100 + 0 = −100.

Exercise 6: Chili Peppers

Problem Description
Ron is cooking chili using an assortment of peppers.

The spiciness of a pepper is measured in Scolville Heat Units (SHU). Ron's chili is currently not spicy at all, but each time Ron adds a pepper, the total spiciness of the chili increases by the SHU value of that pepper.

The SHU values of the peppers available to Ron are shown in the following table:

Pepper Name	Scolville Heat Units
Poblano	1500
Mirasol	6000
Serrano	15500
Cayenne	40000
Thai	75000
Habanero	125000

Your job is to determine the total spiciness of Ron's chili after he has finished adding peppers.

Input Specification
The first line of input will contain a positive integer N, representing the number of peppers Ron adds to his chili. The next N lines will each contain the name of a pepper Ron has added. Each pepper name will exactly match a name that appears in the table above. Note that more than one pepper of the same name can be added.

Output Specification
The output will consist of a positive integer T, representing the total spiciness of Ron's chili.

Sample Input

4
Poblano
Cayenne
Thai
Poblano

Output for Sample Input
118000

Explanation of Output for Sample Input

A Poblano pepper has an SHU value of 1500. A Cayenne pepper has an SHU value of 40000. A Thai pepper has an SHU value of 75000. The total spiciness of Ron's chili is therefore 1500 + 40000 + 75000 + 1500 = 118000.

Exercise 7 : Max
Get two integers from STDIN, compare and print out the larger number.

Exercise 8: **Even or Odd**
Read a number from STDIN, then write a function to determine if the given number is even or odd. If even, print "Not weird", otherwise, print "Weird".

Exercise 9: Grades
Earl of March Secondary School grades its students based on their GPA results as follows:

less than 50 percent	: Fail
50 to 69	: Pass
70 to 79	: Fair
80 to 89	: Good
90 and above	: Excellent

Write a program that gets an integer/GPA score of a student from the keyboard and outputs the appropriate grade.

Exercise 10: Rolling a die
Write a program that simulates a single throw of a die. Use the **random** package to generate an integer from 1 to 6, and ask the user to enter a guess in the range 1 to 6 (print an error message if the input is out of that range). Output to the screen "Correct" if the user correctly guessed the number thrown, otherwise print "Incorrect".

3. Loops

3.1. 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 example: Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

Loop through the letters in the word "banana":

```
for x in "banana":
    print(x)
```

Using the range() function:

```
for x in range(6):
    print(x)
```

Using the start parameter:

```
for x in range(2, 6):
    print(x)
```

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
    print(x)
```

3.2. While loop

With the `while` loop we can execute a set of statements as long as a condition is true.

Example:

```
i = 1
while i < 6:
    print(i)
    i += 1
```

Excercises

Exercise 1: Read an integer n from STDIN, then print all integers up to n in a single line without space.

Sample input:
5

Sample output:
12345

Exercise 2: Reads and integer, n, from STDIN. For all non-negative integers i < n, print i^2.
Example: The list of non-negative integers that are less than 3 is [0, 1, 2]. Print the square of each number on a separate line.
0
1
4

Exercise 3: Sum of cubes
Given a positive integer n, output the sum of cube of all numbers from 1 to n.

Sample Input
4

Sample Output
100

Explanation
100 = $1^3 + 2^3 + 3^3 + 4^3$

Exercise 4: **Fergusonball Ratings**

Fergusonball players are given a star rating based on the number of points that they score and the number of fouls that they commit. Specifically, they are awarded 5 stars for each point scored, and 3 stars are taken away for each foul committed. For every player, the number of points that they score is greater than the number of fouls that they commit.
Your job is to determine how many players on a team have a star rating greater than 40. You also need to determine if the team is considered a gold team which means that all the players have a star rating greater than 40.

Input Specification
The first line of input consists of a positive integer N representing the total number of players on the team. This is followed by a pair of consecutive lines for each player. The first line in a pair is the number of points that the player scored. The second line in a pair is the number of fouls that the player committed. Both the number of points and the number of fouls, are non-negative integers.

Output Specification Output the number of players that have a star rating greater than 40, immediately followed by a plus sign if the team is considered a gold team.

Sample Input 1
3
12
4
10
3
9
1

Output for Sample Input 1
3+

Explanation of Output for Sample Input 1

The image shows the star rating for each player. For example, the star rating for the first player is 12×5−4×3 = 48. All three players have a rating greater than 40 so the team is considered a gold team.

Sample Input 2

```
2
8
0
12
1
```

Output for Sample Input 2

```
1
```

Explanation of Output for Sample Input 2
The image shows the star rating for each player. Since only one of the two players has a rating greater than 40, this team is not considered a gold team.

Exercise 5: Maximum of an array of integers
Given an array of integers, write a program that finds and then prints out the largest number.

Input Specification
The first line of input consists of a positive integer N representing the total number of integers that will be given. Next N lines contain inputs of N integers..

Output Specification Output the maximum integer of the array.

Sample Input 1

```
5
12
14
10
31
9
```

Output for Sample Input 1

```
31
```

Exercise 6: Fibonacci

Write a program to print the first 10 numbers of the Fibonacci sequence.

Hint: F(0) = F(1) = 1, and F(n) = F(n - 1) + F(n - 2)

Exercise 7: Multiples

Write a program that given two numbers a & n, then print out all multiples of a that are smaller or equal to n.

Sample Input
2 15

Sample Output
2 4 6 8 10 12 14

Exercise 8: Sum
Write a program that gets a number n from STDIN, then calculates the sum of the first n positive integers using the for or while loop. Print error if the given number is negative or a decimal.
Hint: S = 1 + 2 + … + n

Sample Input
Enter a number: 10

Sample Output
Sum of the first 10 positive integers is 55

Exercise 9: Primality
Write a program to check whether a given number n is a prime number or not. **Hint**: a number will be prime if it is not divisible by any number that is smaller or equal to its square root. Using for or while loop to scan all numbers from 1 to sqrt(n).

Sample Input:
5

Sample Output:
True

Exercise 10: Pattern

Write a program that asks user for an integer n then prints 2n - 1 lines with the following pattern

```
   +
  +++
 +++++
+++++++
 +++++
  +++
   +
```

Sample input

4

Sample output

```
   +
  +++
 +++++
+++++++
 +++++
  +++
   +
```

5. Function

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

Example:

```
def my_function():
    print("Hello from a function")

my_function()
```

5.1. Function with arguments

```python
def my_function(fname):
    print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

To let a function return a value, use the return statement:

```python
def my_function(x):
    return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

-

5.2. Function With Arbitrary Arguments

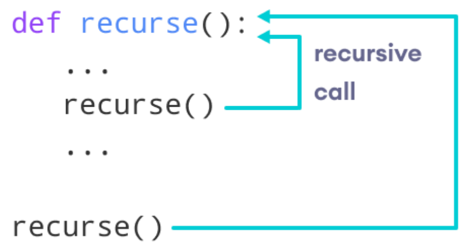
Sometimes, we do not know in advance the number of arguments that will be passed into a function. Python handles this by allowing arbitrary arguments with the use of an asterisk (*). Arbitrary arguments allow us to pass a varying number of values during a function call. For example,

```python
# Return the average of multiple numbers
def average(*arr):
    result = 0

    for x in arr:
        result = result + x

    print("Sum = ", result)
```

```
# function call with 3 arguments
average(3, 5, 4)

# function call with 4 arguments
average(4, 9, 5, 4)
```

5.3. Recursive Function

Recursion is the feature in which a function calls itself. It can be illustrated as in this figure:



```
def recurse():
    ...
    recurse()
    ...

recurse()
```

(**Source**: Python Recursive Function)

The following example uses a recursive function to find the Fibonacci numbers.

```
"""
Recursive function to find the nth term of Fibonacci series
"""
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

Exercises

Exercise 1: **Max**
Write a function **max(a, b)** that takes two parameters a, b, then returns the larger number. The main function will get from STDIN three numbers, call function **max** and output the largest number.

Exercise 2: **Min**
Write a function **min(a, b)** that takes two parameters a, b, then returns the smaller number. The main function will get from STDIN three numbers, call function **min** and output the smallest number.

Exercise 3: Even or Odd
Write a function that checks if a given number is even or odd using the function.

Sample Input:
5

Sample Output:
The entered number is odd.

Exercise 4: Primes
Write a function to check whether a given number n is a prime number or not. Then, print all prime numbers in a given range.

Sample Input:
Enter a range: 4 15

Sample Output:
5, 7, 11, 13

Exercise 5: Perfect square
Write a function to check whether a number is a perfect square or not. **Hint**: a perfect square is a square of an integer.

Sample Input:

25

Sample Output:

The number 25 is a perfect square.

Exercise 6: Fibonacci!
You input a number and the function created checks whether the number belongs to the Fibonacci sequence or not.

Exercise 7: Hamming Distance
Write a function that given two numbers, returns their Hamming distance. **Hint**: Hamming distance is the number of bit positions in which the two bits are different. For example:
a = 1 0 1 0 1 1 1 1 = (175)
b = 1 0 0 0 0 0 0 0 = (128)
Then, the Hamming distance between a and b is 5 as 5 out of 8 bit positions of a and b are different.

Exercise 8: Average of marks
Write a function with arbitrary arguments to calculate the average of marks given from STDIN. The first line of input will be n, the number of marks. The next line will be n marks, separated by a space.

Sample Input
6
70 75 90 80 85 80

Sample Output
80

Exercise 9: Factorial
Write a recursive function that takes an argument n, then return its factorial.

Chapter 3: Math

1. Built-in math functions

Function	**Description**
min(*sequence*)	Return the smallest value in *sequence* .
max(*sequence*)	Return the largest value in *sequence* .
abs(*number*)	Return the absolute value of the argument, | *x* |.
pow(*x*, *y*, [*z*])	Raise *x* to the *y* power. If *z* is present, this is done modulo *z* , x^{y} % *z* .
round(*number*, [*ndigits*])	Round *number* to *ndigits* beyond the decimal point.
cmp(*x*, *y*)	Compare *x* and *y* , returning a number. If the number is less than 0, then *x* < *y* ; if the number is zero, then *x* == *y* ; if the number is positive, then *x* > *y* .
hex(*number*)	Create a hexadecimal string representation of *number* . A leading '0x' is placed on the string as a reminder that this is hexadecimal. hex(684) yields the string '0x2ac'.
oct(*number*)	Create an octal string representation of *number* . A leading '0' is placed on the string as a reminder that this is octal not decimal. oct(509) yields the string '0775'.
bin(*number*)	Create an octal string representation of *number* . A leading '0' is placed on the string as a reminder that this is octal not decimal. oct(509) yields the string '0775'.
int(*string*, [*base*])	Generates an integer from the string *x* . If *base* is supplied, *x* must be in the given base. If *base* is omitted, *x* must be decimal.
str(*object*)	Generate a string representation of the given object. This is a "readable" version of the value.
repr(*object*)	Generate a string representation of the given object. Generally, this is a Python expression that can reconstruct the value; it may be rather long and complex.

2. Math Package

2.2. Functions

Function	Description
math.cos()	Returns the cosine of a number
math.sin()	Returns the sine of a number
math.sqrt()	Returns the square root of a number
math.tan()	Returns the tangent of a number

math.gcd(a, b)	Returns the greatest common divisor of two integers
	
	`import math`
	`print(math.gcd(4, 5))`

2.2. Constants

Constant	Description
math.e	Returns Euler's number (2.7182...)
math.inf	Returns a floating-point positive infinity
math.nan	Returns a floating-point NaN (Not a Number) value
math.pi	Returns PI (3.1415...)
math.tau	Returns tau (6.2831...)

3. Random Package

Function	Description
random.randint(*start*, *stop*)	Return a number between *start* and *stop* (both included):

	`import random`
	`print(random.randint(3, 9))`

random.random()	Returns a random float number between 0 and 1

Exercises

Exercise 1: Given a number, print its representation in binary, octal, and hex

Given an integer, *n*, print the following values for each integer *i* from 1 to *n*:

1. Decimal
2. Octal
3. Hexadecimal (capitalized)
4. Binary

Input Format

A single integer denoting *n*.

Sample Input

17

Sample Output

```
    1     1     1         1
    2     2     2        10
    3     3     3        11
    4     4     4       100
    5     5     5       101
    6     6     6       110
```

7	7	7	111
8	10	8	1000
9	11	9	1001
10	12	A	1010
11	13	B	1011
12	14	C	1100
13	15	D	1101
14	16	E	1110
15	17	F	1111
16	20	10	10000
17	21	11	10001

Exercise 2:
Given an array of numbers. Write a program calculating the sum of all numbers raised to a specific power. Input format: First line contains two integers: n, e separated by a comma. Each of the next n lines contains an integer.

Sample Input:
```
5, 2
1
2
3
4
5
```

Sample Output
```
55
```

Explanation
$55 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2$

Exercise 3:
Write a Python program asking to input a radius R of a sphere and then calculate its surface area and volume. **Hint**: Use math.pi for the value of pi; $A = 4 * pi * R^2$; and $V = 4/3\ pi * R^3$.

Exercise 4:
Given an array of numbers. Write a program calculating the sum of the square root of those numbers. **Input format**: First line contains an integer: n. Each of the next n lines contains an integer.

Sample Input:

```
5
1
4
9
16
25
```

Sample Output

```
15
```

Explanation

15 = sqrt(1) + sqrt(4) + sqrt(9) + sqrt(16) + sqrt(25) = 1 + 2 + 3 + 4 + 5

Exercise 5: **Coprime**

Given n pairs of numbers. Check each pair if it is coprime or not. Print “Coprime” if it is, otherwise print “Not coprime”.

Hint: Use the method math.gcd to find the greatest common divisor of two integers. If it is 1 so two integers are coprime, if not, they are non coprime.

Input format: First line contains an integer: n. Each of the next n lines contains a pair of integers.

Sample Input:

```
5
2 5
4 6
9 15
16 7
11 8
```

Sample Output

```
Coprime
Non coprime
Non coprime
Coprime
Coprime
```

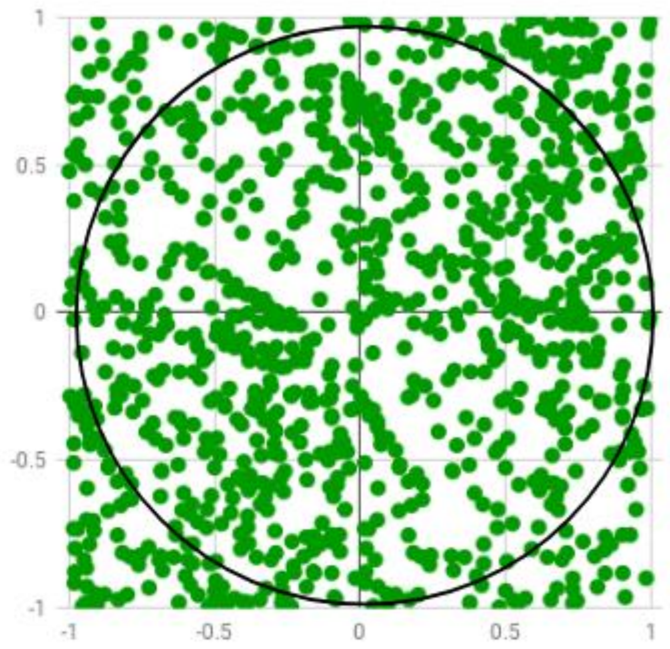
Exercise 6:

Write a program that generates a random number in range [1, 100] and then asks the user to guess (limit 3 times). The program can hint if the number guessed is bigger, lesser the generated random number. The program stops if the user correctly guessed or after three times. The program outputs the random number and the closest guessing number.

Exercise 7: Monte Carlo Method

Using the Monte Carlo method to estimate the value of pi.

<u>Hint</u>: The idea is to simulate random (x, y) points in a 2-D plane with domain as a square of side 2r units centered on (0,0). Imagine a circle inside the same domain with the same radius r and inscribed into the square. We then calculate the ratio of number points that lie inside the circle and total number of generated points.



We know that area of the square is $4r^2$ unit sq while that of circle is pi * r^2 . The ratio of these two areas is as follows :

$$\frac{\text{area of the circle}}{\text{area of the square}} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

Now for a very large number of generated points,

$$\pi = 4 * \frac{\text{no. of points generated inside the circle}}{\text{no. of points generated inside the square}}$$

Chapter 4: Mid-term Projects

Project 1: Rock, Paper, Scissors

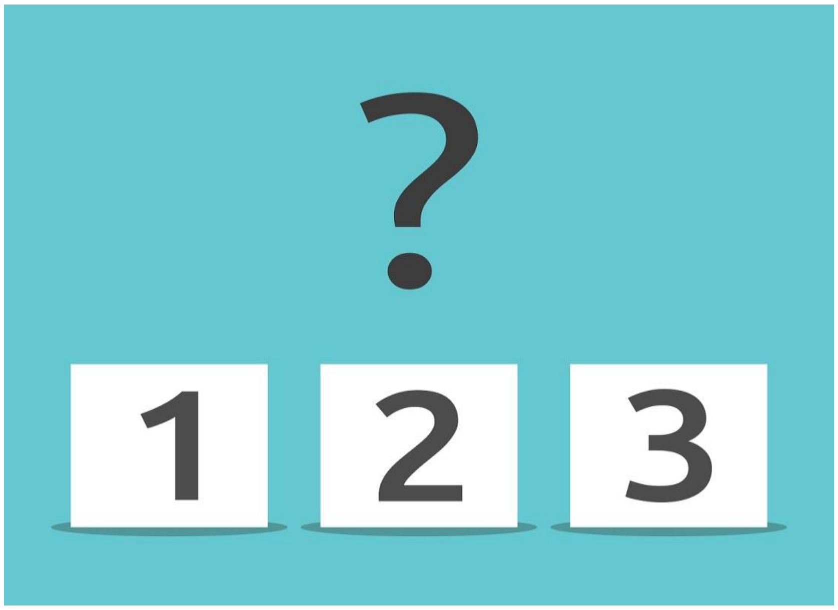
If you are tired of having no playmate, then a 5-minute stint of rock, paper, scissors with the computer and designed by you, yourself will improve your mood.

We again use the random function here. You make a move first and then the program makes one. To indicate the move, you can either use a single alphabet or input an entire string. A function will have to be set up to check the validity of the move.

Using another function, the winner of that round is decided. You can then either give an option of playing again or decide a pre-determined number of moves in advance. A scorekeeping function will also have to be created which will return the winner at the end.

Project 2: Number Guessing

Make a program in which the computer randomly chooses a number between 1 to 10, 1 to 100, or any range. Then give users a hint to guess the number. Every time the user guesses wrong, he gets another clue, and his score gets reduced. The clue can be multiples, divisible, greater or smaller, or a combination of all.



You will also need functions to compare the inputted number with the guessed number, to compute the difference between the two, and to check whether an actual number was inputted or not in this python project. The main aim of this coding project idea from the Python projects list is to introduce beginners to coding basics.

Project 3 Adventure game

This is a basic version of the Adventure game. It is completely text-based. In this version of the game, users can move about through different rooms within a single setting, and based on the user input, it will provide descriptions for each room. This is one of the more interesting python projects for beginners.

Movement direction is crucial here – you must create walls and set the directions in which the users can move through the rooms, set movement restrictions, and also include a tracker that can track how far a user has walked or moved in the game. Mentioning Python projects can help your resume look much more interesting than others.

Project 4 Rolling Dice

As the name of the program suggests, we will be imitating a rolling dice. This is one of the interesting python projects and will generate a random number each dice the program runs, and the users can use the dice repeatedly for as long as he wants. When the user rolls the dice, the program will generate a random number between 1 and 6 (as on a standard dice).

The number will then be displayed to the user. It will also ask users if they would like to roll the dice again. The program should also include a function that can randomly grab a number within 1 to 6 and print it. This beginner-level python project allows you to explore programming fundamentals and different coding concepts.

Chapter 5: Data Structures

1. String

1.1. Declaration

String can be declared either by using single or double quotes:

```
x = "John"
# is the same as
x = 'John'
```

1.2. Basic Operations

Access to a String

A single letter in a string can be accessed via its index (indexed from 0). For example:

```
x = "John"
print(x[2])
# Output
h
```

String Slicing
A substring of a string can be obtained by slicing the index. For example, to get the first 3 letters:

```
x = "Hello John"
print(x[0:2])
# Output
Hel
```

To get the last 4 letters:

```
x = "Hello John"
print(x[-4:])
# Output
John
```

Remark: String is **immutable**, that is, once created, it can not be changed.

```
>>> s = 'Hello'
>>> s[0] = 't'
```

```
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    s[0] = 't'
TypeError: 'str' object does not support item assignment
```

1.3. Methods

Basic functions of strings in Python

Method	Description
mystring[:N]	Extract N number of characters from the start of the string.
mystring[-N:]	Extract N number of characters from end of string
mystring[X:Y]	Extract characters from middle of string, starting from X position and ending with Y

len(string)	Calculate length of string
	`mystring = 'Python'`
	
	`# print the length of mystring`
	`print(len(mystring))`
	
	`# Output: 6`

str.lower()	Convert characters to lowercase
str.upper()	Convert characters to uppercase
str.capitalize()	Converts the first character to uppercase and all other alphabets to lowercase.

str.casefold()	Converts string into lower case
str.title()	Converts the first character of each word to upper case
str.swapcase()	Converts the lower case letters upper case and the upper case letters lower case.
str.contains(`'pattern', case=False`)	Check if pattern matches (Pandas Function)
str.extract(`regular_expression`)	Return matched values (Pandas Function)
str.isalnum()	Check whether string consists of only alphanumeric characters
str.isalpha()	Returns True if all characters in the string are in the alphabet
str.isdecimal()	Returns True if all characters in the string are decimals
str.isdigit()	Returns True if all characters in the string are digits
str.islower()	Check whether characters are all lower case
str.isupper()	Check whether characters are all upper case
str.isnumeric()	Check whether string consists of only numeric characters
str.isspace()	Check whether string consists of only whitespace characters
str.istitle()	Returns True if the string follows the rules of a title
str.isprintable()	Returns True if all characters in the string are printable
str.isidentifier()	Returns True if the string is an identifier
str.isascii()	Returns True if all characters in the string are ascii characters

str.encode()	Returns an encoded version of the string.
	
	`title = 'Python Programming'`
	
	`# change encoding to utf-8`
	`print(title.encode())`
	
	`# Output: b'Python Programming'`

str.format()	Formats specified values in a string
	`# default arguments`
	`print("Hello {}, your balance is {}.".format("Adam", 230.2346))`
	
	`# positional arguments`
	`print("Hello {0}, your balance is {1}.".format("Adam", 230.2346))`

	`# keyword arguments`
	`print("Hello {name}, your balance is {blc}.".format(name="Adam", blc=230.2346))`
	
	`# mixed arguments`
	`print("Hello {0}, your balance is {blc}.".format("Adam", blc=230.2346))`

str.format_map()	Formats specified values in a string. The format_map() method is similar to str.format(**mapping)
	
	`format(**mapping)`
	
	`point = {'x':4,'y':-5}`
	`print('{x} {y}'.format(**point))`
	
	Output
	
	`4 -5`
	
	`format_map(mapping)`
	
	`point = {'x':4,'y':-5}`
	`print('{x} {y}'.format_map(point))`
	
	`point = {'x':4,'y':-5, 'z': 0}`
	`print('{x} {y} {z}'.format_map(point))`
	
	Output
	
	`4 -5`
	`4 -5 0`

str.count(`substring, start=..., end=...`)	Returns the number of times a specified value occurs in a string. `count()` method only requires a single parameter for execution. However, it also has two optional parameters: ● substring - string whose count is to be found. ● start (Optional) - starting index within the string where search starts. ● end (Optional) - ending index within the string where search ends.
str.center(`width, [fillchar]`)	Returns a centered string. The syntax of the `center()` method is: *str.center(width, [fillchar])*

	`s = "Python is great to learn"`
	
	`# returns the centered padded string of length 24`
	`new_string = s.center(30, '*')`
	
	`print(new_string)`
	
	`# Output: ***Python is great to learn****`

str.find(`sub[, start[,end]]`)	Return position of substring or pattern. The `find()` method takes maximum of three parameters: ● sub - It is the substring to be searched in the `str` string. ● start and end (optional) - The range `str[start:end]` within which substring is searched. If the substring exists inside the string, it returns the index of the first occurrence of the substring. If not, it returns -1.
str.rfind(`sub[, start[, end]]`)	`rfind()` method takes a maximum of three parameters: ● sub - It's the substring to be searched in the `str` string. ● start and end (optional) - substring is searched within `str[start:end]` If substring exists inside the string, it returns the highest index where substring is found. If not, returns -1.
str.index()	Searches the string for a specified value and returns the position of where it was found
str.rindex()	Searches the string for a specified value and returns the last position of where it was found
str.rjust()	Returns a right justified version of the string
str.ljust()	Returns a left justified version of the string

str.strip(`[chars]`)	Returns a trimmed version of the string after removing characters `[chars]` from both left and right sides.
	
	`random_string = '   this is good '`
	
	`# Leading whitespace are removed`
	`print(random_string.strip())`

	`# Argument doesn't contain space`
	`# No characters are removed.`
	`print(random_string.strip('sti'))`
	
	`print(random_string.strip('d to'))`
	
	`website = 'https://www.programiz.com/'`
	`print(website.strip('htps:/.'))`
	
	`# Output`
	
	`this is good`
	`   this is good`
	`his is g`
	`www.programiz.com`

str.lstrip(`[chars]`)	Returns a left trim version of the string after removing `[chars]` from left side
str.rstrip(`[chars]`)	Returns a left trim version of the string after removing `[chars]` from right side

`separator`.join()	Converts the elements of an iterable into a string
	`text = ['Python', 'is', 'a', 'fun', 'programming', 'language']`
	
	`# join elements of text with space`
	`print(' '.join(text))`
	
	`# Output: Python is a fun programming language`

str.split(`separator[, maxsplit]`)	Splits a string at the specified separator and returns a list of substrings. The split() method takes a maximum of 2 parameters:
	● separator (optional)- Delimiter at which splits occur. If not provided, the string is splitted at whitespaces.
	● maxsplit (optional) - Maximum number of splits. If not provided, there is no limit on the number of splits.
	
	`random_string = 'Python is awesome'`
	
	`# Splitting based on whitespace`
	`print(random_string.split())`
	
	`# Output: [‘Python’, ‘is’, ‘awesome’]`
	
	`random_string = 'Study-Python-is fun'`

	`# Splitting based on whitespace`
	`print(random_string.split('-', 1))`
	
	`# Output: ['Study', 'Python-is fun']`

str.rsplit(`separator[, maxsplit]`)	Splits the string at the specified separator, and returns a list of substrings. The split() method takes a maximum of 2 parameters: ● separator (optional)- Delimiter at which splits occur. If not provided, the string is splitted at whitespaces. ● maxsplit (optional) - Maximum number of splits. If not provided, there is no limit on the number of splits.
str.replace(`old_substring, new_substring`)	Replace a part of text with different substring
separator.join(str)	Concatenate Strings

Exercises

Exercise 1: Get a sentence S from STDIN, print the following lines:
- Line 1: print the number of words in S and length of the string S
- Line 2: print first 5 character of S
- Line 3: print last 2 character of S
- Line 4: print characters between positions 2 and 8
- Line 5: print characters at odd positions
- Line 6: print S in reversed order
- Line 7: print S in UPPERCASE
- Line 8: print S in LOWERCASE
- Line 9: print 3rd character of S
- Print all characters in S in separate lines.

Exercise 2: Get two sentences S1 and S2 from STDIN. Print the following lines into the STDOUT:
- Line 1: print *True* if two sentences are identical, otherwise, print *False*.
- Line 2: print the concatenation of S1 and S2

Exercise 3: You are given a string and your task is to swap cases. In other words, convert all lowercase letters to uppercase letters and vice versa.

For Example:

Www.HackerRank.com → wWW.hACKERrANK.COM

Pythonist 2 → pYTHONIST 2

Exercise 4: You are given a string. Split the string on a " " (space) delimiter and join using a - hyphen.
Sample Input
this is a string

Sample Output
this-is-a-string

Exercise 5: You are given the firstname and lastname of a person on two different lines. Your task is to read them and print the following:

Hello *firstname lastname*! You just delved into python.

Exercise 6: Read a given string, change the character at a given index and then print the modified string.
Input Format
The first line contains a string *S*.
The next line contains an integer *position*, the index location and a string *character*, separated by a space.

Sample Input
abracadabra
5 k

Sample Output
abrackdabra

Exercise 7:
In this challenge, the user enters a string and a substring. You have to print the number of times that the substring occurs in the given string. String traversal will take place from left to right, not from right to left.

NOTE: String letters are case-sensitive.

Input Format

The first line of input contains the original string. The next line contains the substring.

Output Format

Output the integer number indicating the total number of occurrences of the substring in the original string.

Sample Input

```
ABCDCDC
CDC
```

Sample Output

```
2
```

Exercise 8: You are given a string. Your task is to find out if the string contains: alphanumeric characters, alphabetical characters, digits, lowercase and uppercase characters.

Input Format

A single line containing a string *S*.

Output Format

In the first line, print True if *S* has any alphanumeric characters. Otherwise, print *False*.

In the second line, print True if *S* has any alphabetical characters. Otherwise, print *False*.

In the third line, print True if *S* has any digits. Otherwise, print *False*.

In the fourth line, print True if *S* has any lowercase characters. Otherwise, print *False*.

In the fifth line, print True if *S* has any uppercase characters. Otherwise, print *False*.

Sample Input

```
qA2
```

Sample Output

```
True
True
True
True
True
```

Exercise 9a: Textwrap

You are given a string *S* and width *w*. Your task is to wrap the string into a paragraph of width *w*.

Input Format

The first line contains a string, .

The second line contains the width, .

Sample Input

```
ABCDEFGHIJKLIMNOQRSTUVWXYZ
4
```

Sample Output

```
ABCD
EFGH
IJKL
IMNO
QRST
UVWX
YZ
```

Exercise 9b: Textwrap 2

You are given a string *S* of length *n* and width *w*, where *n* is multiple of *w*, and s = $c_0c_1c_2\ldots c_{n-1}$. Split s into n/w substrings, then transform these substrings into substrings without repeated characters. For example, s = ‘AAABCADDE’, w = 3. Output ‘A’, ‘BCA’, and ‘DE’.

Explanation:

There are three substrings of length 3 to consider: 'AAA', 'BCA' and 'DDE'. The first substring is all 'A' characters, so return only ‘A’. The second substring has all distinct characters, so return ‘BCA’. The third substring has different characters, so return ‘DE’. Note that a subsequence maintains the original order of characters encountered. The order of characters in each subsequence shown is important.

Sample Input

AABCAAADA

3

Sample Output

AB

CA

AD

Exercise 10: **Capitalize Problem**

You are asked to ensure that the first and last names of people begin with a capital letter in their passports. For example, alison heck should be capitalised correctly as Alison Heck.

Given a full name, your task is to capitalize the name appropriately.

Input Format

A single line of input containing the full name, .

Output Format

Print the capitalized string, .

Sample Input

chris alan

Sample Output

Chris Alan

Exercise 11: **Special Event**
Problem Description
You are trying to schedule a special event on one of five possible days. Your job is to determine on which day you should schedule the event, so that the largest number of interested people are able to attend.

Input Specification
The first line of input will contain a positive integer N, representing the number of people interested in attending your event. The next N lines will each contain one person's availability using one character for each of Day 1, Day 2, Day 3, Day 4, and Day 5 (in that order). The character Y means the person is able to attend and a period (.) means the person is not able to attend.

The following table shows how the available 15 marks are distributed:

Marks	**Description**
6	There will be exactly one day on which every person will be able to attend.
6	There will be exactly one day on which the largest number of people will be able to attend.
3	There might be more than one day on which the largest number of people will be able to attend.

Output Specification
The output will consist of one line listing the day number(s) on which the largest number of interested people are able to attend.
If there is more than one day on which the largest number of people are able to attend, output all of these day numbers in increasing order and separated by commas (without spaces).

Sample Input 1
```
3
YY. Y.
. . . Y .
 .YYY.
```

Output for Sample Input 1
```
4
```
Explanation of Output for Sample Input 1 All three people are able to attend on Day 4, and they are not all available on any other day.

Exercise 12: **Reversed string**
Given two strings s1 and s2, write a program to check if s2 is a reversed string of s1. For example ‘olleH’ is the reversed string of ‘Hello’.

Exercise 13: **Anagram**
Write a program to check whether two given strings are an anagram. **Hint**: two words/strings are anagram if one can get from another by scrambling letters. For example: *spare* and *pears* are anagram.

2. List

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

2.1. Declaration

Lists are used to store multiple items in a single variable. Lists are created using square brackets:

Example:
Create a List:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

List items can be of any data type:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
list4 = ["abc", 34, True, 40, "male"]
```

2.2. Methods

The following methods for list in Python

Method	**Description**
append(*item*)	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count(*value*)	Returns the number of elements with the specified value
extend(*elements*)	Add the elements of a list (or any iterable), to the end of the current list
index(*value*)	Returns the index of the first element with the specified value
insert(*position, value*)	Adds an element at the specified position
pop(*position*)	Removes the element at the specified position
remove(*value*)	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list
print(*list*)	Print the list

Exercises

Exercise 1:
Given a list, write a program that print the reversed list.

Exercise 2:
Given two lists of equal numbers of integers. Write a program that return a list of sum of integers at the same positions of the two given lists.

Sample Input:
3, 4, 7, 2
5, 2, 7, 1

Sample Output:
[8, 6, 14, 3]

Exercise 3:
Given a list of integers, write a program that returns a list of cubes of items in the list.

Sample Input:
List1 = [3, 4, 7, 2]

Sample Output:
[27, 64, 343, 8]

Exercise 4:
Given a list of strings, write a program removing the empty string in the list.

Sample input
[“Micheal”, “Mary”, “”, “Benjamin”, “”, “Brian”]

Sample output
[“Micheal”, “Mary”, “Benjamin”, “Brian”]

Exercise 5: Initialize your list and read in the value n followed by n lines of commands from STDIN where each command will be of the list methods listed above. Iterate through each command in order and perform the corresponding operation on your list.
Example
N = 4
append 1
append 2
insert 3 1
print

- append 1: Append 1 to the list, arr = [1].
- append 2: Append 2 to the list, arr = [1, 2].
- insert 3 1: Insert 3 at index 1, arr = [1, 3, 2].
- print: Print the array.

Output:
[1, 3, 2]

Sample Input 0

```
12
insert 0 5
insert 1 10
insert 0 6
print
remove 6
append 9
append 1
sort
print
pop
reverse
print
```

Sample Output 0

```
[6, 5, 10]
[1, 5, 9, 10]
[9, 5, 1]
```

Exercise 6: You are given three integers x, y and z, representing the dimensions of a cuboid along with an integer n. Print a list of all possible coordinates given by (i, j, k) on a 3D grid where the sum of i + j + k is not equal to n. Here, 0 <= i <= x, 0 <= j <= y, 0 <= k <= z. Please use list comprehensions rather than multiple loops, as a learning exercise.

Input Format

Four integers x, y, z and n, each on a separate line.

Constraints

Print the list in lexicographic increasing order.

Sample Input 0

```
1
1
1
2
```

Sample Output 0

```
[[0, 0, 0], [0, 0, 1], [0, 1, 0], [1, 0, 0], [1, 1, 1]]
```

Explanation 0

Each variable x, y and z will have values of 0 or 1. All permutations of lists in the form.
[[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]
Remove all arrays that sum to to leave only the valid permutations.

Sample Input 1

```
2
2
2
2
```

Sample Output 1

```
[[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 2], [0, 2, 1], [0, 2, 2], [1, 0, 0], [1, 0, 2], [1, 1, 1], [1, 1, 2], [1, 2, 0], [1,
2, 1], [1, 2, 2], [2, 0, 1], [2, 0, 2], [2, 1, 0], [2, 1, 1], [2, 1, 2], [2, 2, 0], [2, 2, 1], [2, 2, 2]]
```

3. Sets

A set is a collection of unique data. That is, elements of a set cannot be duplicated. For example, suppose we want to store information about *student IDs*. Since *student IDs* cannot be duplicated, we can use a set.

When printed, iterated or converted into a sequence, its elements will appear in an arbitrary order.

3.1. Create a Set in Python

In Python, we create sets by placing all the elements inside curly braces {}, separated by comma. A set can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements. Let's see an example,

```
# create a set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)

# create a set of string type
vowel_letters = {'a', 'e', 'i', 'o', 'u'}
print('Vowel Letters:', vowel_letters)

# create a set of mixed data types
mixed_set = {'Hello', 101, -2, 'Bye'}
print('Set of mixed data types:', mixed_set)
```

3.2. Methods for Set in Python

Method	**Description**
add(*value*)	Add an item *value* into the set

update(*items*)	update the set with items other collection types (lists, tuples, sets, etc). For example,
	
	`companies = {'Lacoste', 'Ralph Lauren'}`
	`tech_companies = ['apple', 'google', 'apple']`
	
	`companies.update(tech_companies)`
	
	`print(companies)`

Method	**Description**
	`# Output: {'google', 'apple', 'Lacoste', 'Ralph Lauren'}`

discard(*value*)	Remove an item *value* from the set. For example
	`languages = {'Swift', 'Java', 'Python'}`
	
	`print('Initial Set:',languages)`
	
	`# remove 'Java' from a set`
	`removedValue = languages.discard('Java')`
	
	`print('Set after remove():', languages)`
	Output
	
	`Initial Set: {'Python', 'Swift', 'Java'}`
	`Set after remove(): {'Python', 'Swift'}`

all()	Returns `True` if all elements of the set are true (or if the set is empty).
any()	Returns `True` if any element of the set is true. If the set is empty, returns `False`.
enumerate()	Returns an enumerate object. It contains the index and value for all the items of the set as a pair.
len()	Returns the length (the number of items) in the set.
max()	Returns the largest item in the set.
min()	Returns the smallest item in the set.
sorted()	Returns a new sorted list from elements in the set(does not sort the set itself).
sum()	Returns the sum of all elements in the set.
A.union(B)	The union of two sets A and B include all the elements of set A and B.
A.intersection(B) Or A & B	In Python, we use the & operator or the intersection() method to perform the set intersection operation.

Method	Description
A.difference(B)	The difference between two sets A and B include elements of set A that are not present on set B.
A.symmetric_difference(B)	The symmetric difference between two sets A and B includes all elements of A and B without the common elements.

Exercises

Exercise 1: Ms. Gabriel Williams is a botany professor at District College. One day, she asked her student Mickey to compute the average of all the plants with distinct heights in her greenhouse.

Formula used:

Average = sum of distinct heights / total number of distinct heights

Input Format

The first line contains the integer, N, the size of arr.

The second line contains the N space-separated integers, arr[i].

Sample Input

```
10
161 182 161 154 176 170 167 171 170 174
```

Sample Output

```
169.375
```

Explanation

Here, set {161 182 154 176 167 171 170 174} is the set containing the distinct heights. Using the sum() and len() functions, we can compute the average.

Exercise 2: Given sets of integers, and , print their symmetric difference in ascending order. The term symmetric difference indicates those values that exist in either or but do not exist in both.

Input Format

The first line of input contains an integer, .

The second line contains space-separated integers.

The third line contains an integer, .

The fourth line contains space-separated integers.

Output Format

Output the symmetric difference integers in ascending order, one per line.

Sample Input

4

2 4 5 9

4

2 4 11 12

Sample Output

5

9

11

12

Exercise 3: There is an array arr of *n* integers. There are also *2* **disjoint sets**, A and B, each containing *m* integers. You like all the integers in set A and dislike all the integers in set B. Your initial happiness is 0. For each *i* integer in the array arr, if i in A, you add 1 to your happiness. If i in B, you minus 1 to your happiness. Otherwise, your happiness does not change. Output your final happiness at the end.

Note: Since A and B are sets, they have no repeated elements. However, the array might contain duplicate elements.

Input Format

The first line contains integers n and m separated by a space.

The second line contains n integers, the elements of the array arr.

The third and fourth lines contain m integers, A and B, respectively.

Output Format

Output a single integer, your total happiness.

Sample Input

```
3 2
1 5 3
3 1
5 7
```

Sample Output

```
1
```

Explanation

In set A, you gain 1 unit of happiness for elements 1 and 3. You lose 1 unit for 5 in set B. The element 7 in set B does not exist in the array so it is not included in the calculation.

Hence, the total happiness is 1.

Exercise 4: You have a non-empty set *s*, and you have to execute *N* commands given in *N* lines.

The commands will be *pop*, *remove* and *discard*.

Input Format

The first line contains integer *n*, the number of elements in the set *s*.

The second line contains *n* space separated elements of set *s*. All of the elements are non-negative integers, less than or equal to 9.

The third line contains integer *N*, the number of commands.

The next *N* lines contain either *pop*, *remove* and/or *discard* commands followed by their associated value.

Output Format

Print the sum of the elements of set *s* on a single line.

Sample Input

```
9
1 2 3 4 5 6 7 8 9
10
pop
remove 9
discard 9
discard 8
remove 7
pop
discard 6
remove 5
pop
discard 5
```

Sample Output

```
4
```

Explanation

After completing these 10 operations on the set, we get set s = ([4]). Hence, the sum is 4.

Exercise 5: The students of District College have subscriptions to English and French newspapers. Some students have subscribed only to English, some have subscribed to only French and some have subscribed to both newspapers.

You are given two sets of student roll numbers. One set has subscribed to the English newspaper, and the other set is subscribed to the French newspaper. The same student could be in both sets.

Your task is to find the total number of students who have subscribed to at least one newspaper and the total number of students who have subscribed to both newspapers.

Input Format

The first line contains an integer, *n*, the number of students who have subscribed to the English newspaper.
The second line contains *n* space separated roll numbers of those students.
The third line contains *b*, the number of students who have subscribed to the French newspaper.
The fourth line contains *b* space separated roll numbers of those students.

Output Format

First line: Output the total number of students who have at least one subscription.

Second line: Output the total number of students who have subscriptions to **both** English and French newspapers.

Third line: Output the total number of students who are subscribed to the English newspaper only.

Fourth line: Output total number of students who have subscriptions to the English or the French newspaper but not both.

Sample Input

```
9
1 2 3 4 5 6 7 8 9
9
10 1 2 3 11 21 55 6 8
```

Sample Output

```
13
5
4
8
```

Explanation

Roll numbers of students who have at least one subscription:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21 and 55. Roll numbers: 1, 2, 3, 6 and 8 are in both sets so they are only counted once.

Hence, the total is 13 students and the number of students registered is 5. The students who subscripted only to English newspapers are 4, 5, 7, 9, so the returned result is 4. The students who subscripted only to English or French newspapers, but not both, are 4, 5, 7, 9, 10, 11, 21, 55 so the returned result is 8.

Exercise 6: You are given a set A and N number of other sets. These N numbers of sets have to perform some specific mutation operations on set A. Your task is to execute those operations and print the sum of elements from set A.

Input Format

The first line contains the number of elements in set A.

The second line contains the space separated list of elements in set A.

The third line contains integer N, the number of other sets.

The next *2*N* lines are divided into parts containing two lines each.

The first line of each part contains the space separated entries of the operation name and the length of the other set.

The second line of each part contains space separated list of elements in the other set.

Output Format
Output the sum of elements in set A.

Sample Input

```
16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 24 52
4
intersection_update 10
2 3 5 6 8 9 1 4 7 11
update 2
55 66
symmetric_difference_update 5
22 7 35 62 58
difference_update 7
11 22 35 55 58 62 66
```

Sample Output

```
38
```

Exercise 7: Given a list of numbers from STDIN, print the number of occurrences of items in this list.

Sample Input

```
1 2 3 6 3 4 4 4 5 3 6 1 6 5 3 2 4 1 2 5 1 4 3 6 8 4 3 1 3 6 2
```

Sample Output

```
1 5
2 4
3 7
4 6
5 3
6 5
8 1
```

4. Dictionary

Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is ordered, changeable and do not allow duplicates.

4.1. Methods for Dictionary in Python

Python has a set of built-in methods that you can use on dictionaries.

Method	**Description**
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair

keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Exercises

Exercise 1: Ordering dictionary

Given a dictionary, write a program to sort the dictionary by keys and the result type will be a dictionary.

`Sample Input:`
{'ravi': 10, 'rajnish': 9, 'sanjeev': 15, 'yash': 2, 'suraj': 32}

`Sample Output:`
{'rajnish': 9, 'ravi': 10, 'sanjeev': 15, 'suraj': 32, 'yash': 2}

Exercise 2: Combination

Write a Python program to combine two dictionaries by adding values for common keys.

Sample Input:
dict1 = {'a': 100, 'b': 200, 'c':300}
dict2 = {'a': 300, 'b': 200, 'd':400}

Sample output:
dict3 = {'a': 400, 'b': 400, 'c': 300, 'd': 400}

Exercise 3: Sum

Write a program to find the sum of all items in the dictionary.

Sample Input:
dict = {'a': 100, 'b': 200, 'c':300}

Sample output:
600

Exercise 4:
Write a Python program to print all distinct values in a dictionary

Sample Input:

{"1":"abc", "2": "abcd", "3": "abcd", "4": "def", "5":"def", "1":"dbfc", "7":"abc"}

Sample Output:
'abc', 'abcd', 'def', ‘dbfc'

Exercise 5:
Write a Python program to combine two lists into a dictionary

Sample Input:

keys = [1, 2, 3, 4, 5]
values = [‘one’, ‘two’, ‘three’, ‘four’, ‘five’]

Sample Output:
{1:’one’, 2: ‘two’, 3: ‘three’, 4: ‘four’, 5:’five’}