NMK40403

Artificial Intelligence

Lab 2: Python Programming

Mohamed Elshaikh
UNIMAP - FKTEN

1 Objectives

- Python Variables
- Python Data types
- Python Lists

1.1 Programming Python Variables

In programming, variables serve as essential tools for storing, managing, and manipulating data. In Python, variables act as containers that hold information, making it easy to access and modify values throughout a program. This laboratory focuses on the foundational concept of variables in Python, covering how to create, assign, and work with various data types. You will explore the rules of naming variables, understand the significance of data types (such as integers, floats, strings, and booleans), and practice basic operations with these variables. Mastering variable management is a crucial first step toward building complex and functional Python programs, laying the groundwork for developing more advanced coding skills in future labs.

1.2 Saving Values

In this part, we'll learn how to save values in Python, how to update values, and how to use syntax shortcuts to update values. Understanding how to work with values is a foundation of programming and is used for tasks ranging from simple calculations to complex data analysis and AI algorithms. Throughout this lesson, we'll learn core Python programming concepts in the context of real data. We'll be using data from the table below, which provides some information about five mobile applications from the iOS store:

track_name	price	currency	rating_count_tot	user_rating
Facebook	0.0	USD	2974676	3.5
Instagram	0.0	USD	2161558	4.5
Clash of Clans	0.0	USD	2130805	4.5
Fruit Ninja Classic	1.99	USD	698516	4.5
Minecraft: Pocket Edition	6.99	USD	522012	4.5

Let's say we want to save the result of an arithmetic operation. For example, the cost of buying Fruit Ninja Classic for two family members, 1.99 * 2 equals 3.98, and we want to save 3.98. This is the code we need to run to save 3.98:

```
result = 3.98
```

If we print the name result, the output is 3.98:

```
result = 3.98
print(result)
```

```
Output
3.98
```

We can also directly save 1.99 * 2 instead of saving 3.98.

```
result = 1.99 * 2
print(result)
```

```
Output
3.98
```

Notice, however, that print(result) outputs 3.98, not 1.99 * 2. This is because the computer first calculates 1.99 * 2 and then saves the result 3.98 to result.

1.2.1 Example

Use python code to show the cost of buying Minecraft: Pocket Edition for two family members.

Solution

1- Save the result of 6.99 * 2 to result.
2- Print result.

```
result = 3.98
print(result)
result = 1.99 * 2
print(result)
result = 6.99 * 2
print(result)
```

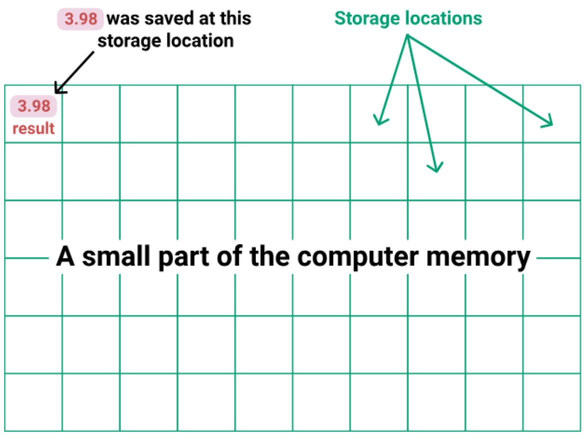
What will be the output of the above?

1.3 Variables

In the last example, we saved 3.98 to result. Let's learn more about that.

```
result = 3.98
```

When we run this code, 3.98 gets stored in a specific location in your computer's memory. Each storage location in the computer's memory has a unique label or identifier, and in this case, the label is result. This label helps us find and use the stored value (3.98) whenever we need it.


3.98 was saved at this storage location
Storage locations
3.98
result
A small part of the computer memory

We can use the result label to get our 3.98 value and use it in other parts of our code:

```
result = 3.98
print(result)
print(result + 1.99)
```

```
Output
3.98
5.97
```

This labeled storage location, result, is what we call a variable. The name result is a variable name. We need to remember to write the variable name on the left side of the = sign, and the value we want to store on the right side. So to store 3.98 in a variable named result, we write result = 3.98, not 3.98 = result. The name result was our choice, but we could have used any other name:

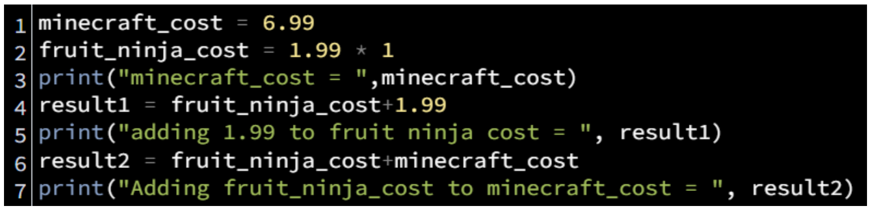
```
fruit_ninjas = 3.98
print(fruit_ninjas)
print(fruit_ninjas + 1.99)
```

```
Output
3.98
5.97
```

1.3.1 Example

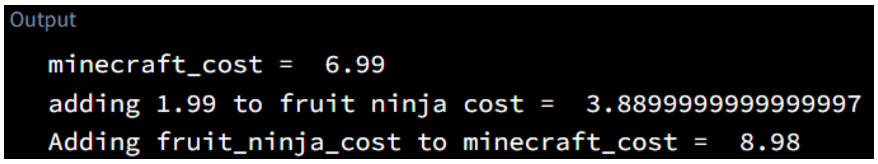
a. Store the value 6.99 in a variable named minecraft_cost.
b. Store the result of 1.99 * 1 to a variable named fruit_ninja_cost.
c. Using the print() function, display the following:
   - The value stored in the minecraft_cost variable.
   - The result of adding 1.99 to the variable fruit_ninja_cost.
   - The result of adding fruit_ninja_cost to minecraft_cost.

Solution



```
minecraft_cost = 6.99
fruit_ninja_cost = 1.99 * 1
print("minecraft_cost = ",minecraft_cost)
result1 = fruit_ninja_cost+1.99
print("adding 1.99 to fruit ninja cost = ", result1)
result2 = fruit_ninja_cost+minecraft_cost
print("Adding fruit_ninja_cost to minecraft_cost = ", result2)
```

Output:



```
Output
  minecraft_cost =  6.99
  adding 1.99 to fruit ninja cost =  3.8899999999999997
  Adding fruit_ninja_cost to minecraft_cost =  8.98
```

1.4 Variable Names

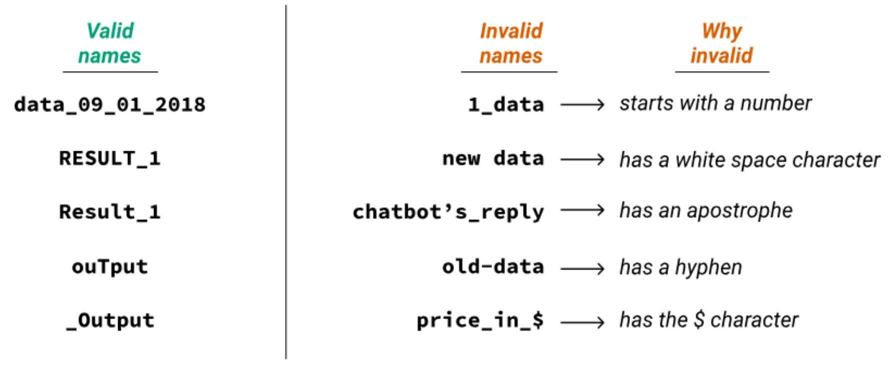
On the previous examples, we learned that we can choose different names for variables. However, these names must follow a number of syntax rules. For instance, naming a variable a result will output a syntax error because we can't use space characters in variable names.

```
a result = 3.98
```

```
Output
    a result = 3.98
       ^
SyntaxError: invalid syntax
```

These are the two syntax rules we need to follow when we're naming variables:

1. We must use only letters, numbers, or underscores (we can't use apostrophes, hyphens, spaces, etc.).
2. Variable names can't begin with a number.



Valid names	*Invalid names*	*Why invalid*
`data_09_01_2018`	`1_data`	*starts with a number*
`RESULT_1`	`new data`	*has a white space character*
`Result_1`	`chatbot’s_reply`	*has an apostrophe*
`ouTput`	`old-data`	*has a hyphen*
`_Output`	`price_in_$`	*has the $ character*

Note that variable names are case sensitive, which means that a variable named result is different than a variable named Result:

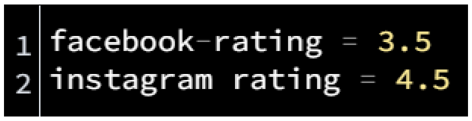
```
result = 3.98
Result = 6.99
print(result)
print(Result)
```

```
Output
3.98
6.99
```

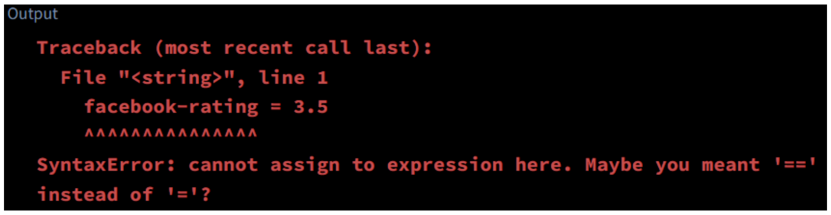
1.4.1 Example

Write a python code to store 3.5 in a variable named facebook-rating and 4.5 in a variable named instagram rating. But both of these variable names cause syntax errors, so we need to fix them.

1- Change the variable name facebook-rating to facebook_rating to prevent a syntax error.
2- Change the variable name instagram rating to instagram_rating to prevent a syntax error.

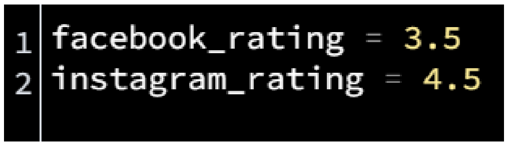


```
facebook-rating = 3.5
instagram rating = 4.5
```



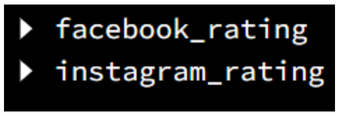
```
Output
  Traceback (most recent call last):
    File "<string>", line 1
      facebook-rating = 3.5
      ^^^^^^^^^^^^^^^
  SyntaxError: cannot assign to expression here. Maybe you meant '=='
  instead of '='?
```

To resolve:



```python
facebook_rating = 3.5
instagram_rating = 4.5
```

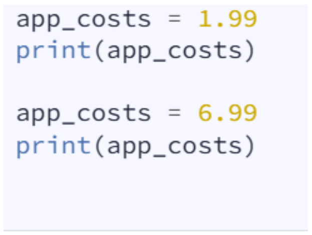
After run:



```
▸ facebook_rating
▸ instagram_rating
```

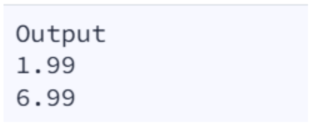
1.5 Updating Variables

We can update the value stored in a variable. Below, we first store 1.99 in the variable app_costs, and then we update app_costs to store 6.99 instead.



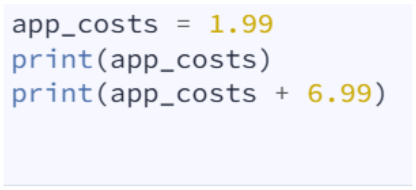
```python
app_costs = 1.99
print(app_costs)

app_costs = 6.99
print(app_costs)
```

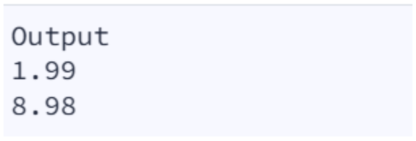


```
Output
1.99
6.99
```

We can also update a variable by doing arithmetic operations:



```python
app_costs = 1.99
print(app_costs)
print(app_costs + 6.99)
```



```
Output
1.99
8.98
```

Above, the variable app_costs initially stores a value of 1.99. Then, app_costs + 6.99 evaluates to 8.98 because app_costs stores a value of 1.99, so app_costs + 6.99 becomes 1.99 + 6.99. Let's look at another example:

```
app_costs = 1.99
app_costs = app_costs + 6.99
print(app_costs)
```

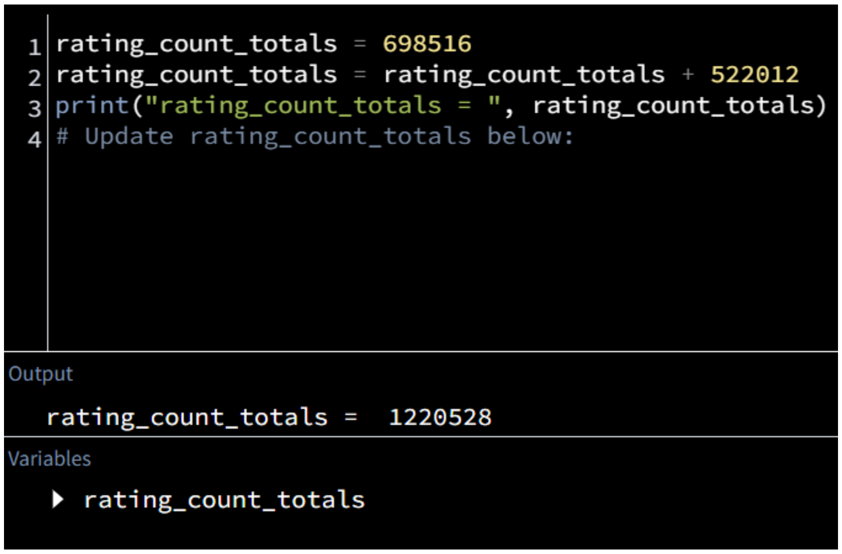
```
Output
8.98
```

Notice above that when we run app_costs = app_costs + 6.99, app_costs updates to store the result of app_costs + 6.99, which is 8.98. Running app_costs = app_costs + 6.99 is the same as running app_costs = 1.99 + 6.99 because app_costs stores 1.99. Now let's practice updating variables.

1.5.1 Example

The rating count total of Fruit Ninja Classic is 698516, and the rating count total of Minecraft: Pocket Edition is 522012. Let's use these values in this exercise.

a. Update the variable rating_count_totals by adding 522012 to its current value.
b. Print rating_count_totals.



```
rating_count_totals = 698516
rating_count_totals = rating_count_totals + 522012
print("rating_count_totals = ", rating_count_totals)
# Update rating_count_totals below:
```

Output

```
rating_count_totals =  1220528
```

Variables

```
▶ rating_count_totals
```

1.6 Syntax Shortcuts

On the previous example, we used the code app_costs = app_costs + 6.99 to update app_costs from 1.99 to 8.98.

```
app_costs = 1.99
print(app_costs)
print(app_costs + 6.99)

app_costs = app_costs + 6.99
print(app_costs)
```

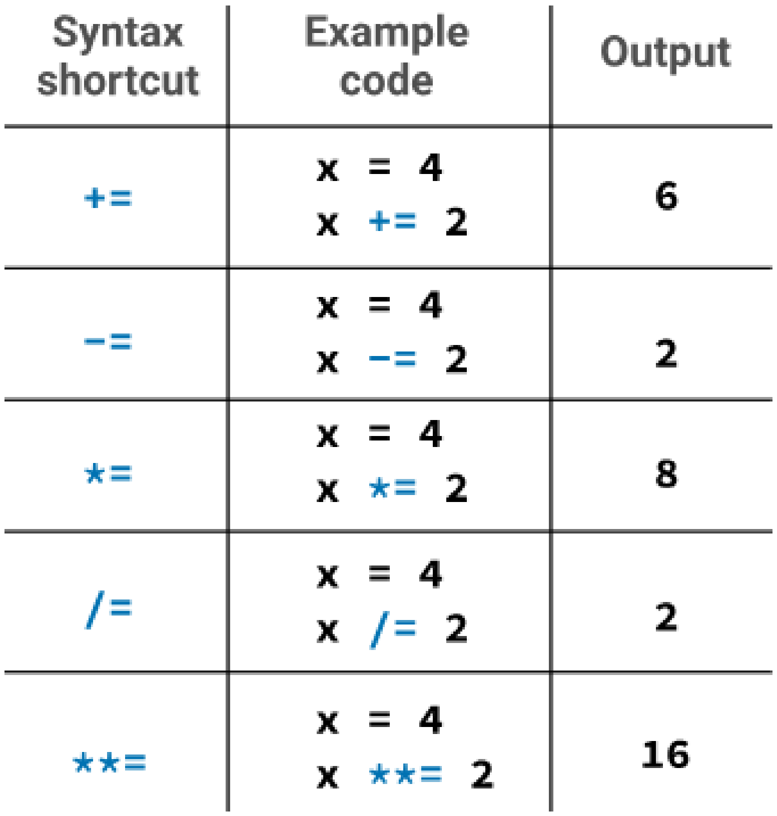
```
Output
1.99
8.98
8.98
```

There are several syntax shortcuts we can use to update a variable when we're doing arithmetic operations. For example, as an alternative to the code above, we can write app_costs += 6.99 (instead of app_costs = app_costs + 6.99):

```
app_costs = 1.99
app_costs += 6.99
print(app_costs)
```

```
Output
8.98
```

Below is a table with some syntax shortcuts we can use:



Syntax shortcut	Example code	Output

+=	x = 4
	x += 2
		6

-=	x = 4
	x -= 2
		2

*=	x = 4 x *= 2	8
/=	x = 4 x /= 2	2
**=	x = 4 x **= 2	16

Note that we can only use these operators (+=, -=, *=, /=, **=) to update a variable. This means the variable we're updating must already store a value. In other words, the variable must already be defined. When we try to update a variable that we haven't defined, we get an error called NameError.

```
# Error:
app_costs += 6.99
```

```
Output
NameError: name 'app_costs' is not defined
```

```
# No Error:
app_costs = 1.99
app_costs += 6.99
print(app_costs)
```

```
# No Error:
app_costs = 1.99
app_costs += 6.99
print(app_costs)
```

```
Output
8.98
```

This kind of error is different from a syntax error. app_costs += 6.99 is correct Python syntax, but the computer returns an error because it can't update a variable that hasn't been defined. Whenever the syntax is correct but the computer still returns an error for one reason or another, we get a runtime error.

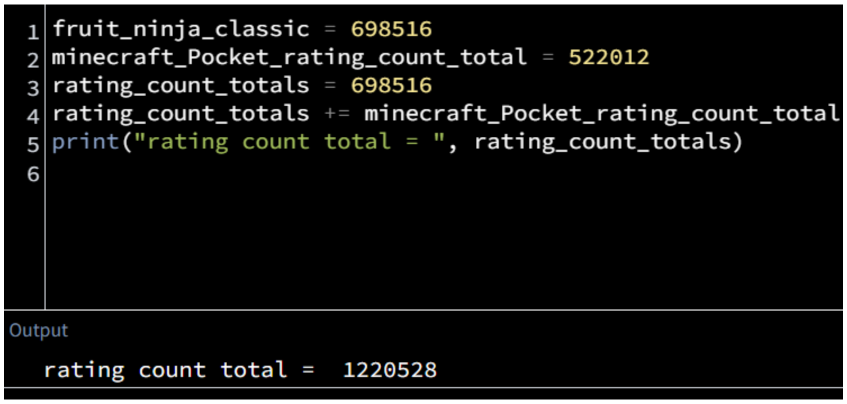
Notice also that we updated a variable using app_costs = app_costs + 6.99. In mathematics, app_costs = app_costs + 6.99 would be a false statement because app_costs can never be equal to app_costs + 6.99. This tells us that the = operator doesn't have the same meaning as it does in mathematics.

In Python, the = operator tells us that the value on the right is assigned to the variable on the left. It doesn't tell us anything about equality. We call = an assignment operator, and we read code like x = 5 as "five is assigned to x" or "x is assigned five," not "x equals five."

1.6.1 Example

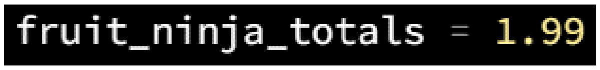
The rating count total of Fruit Ninja Classic is 698516, and the rating count total of Minecraft: Pocket Edition is 522012. The cost of Fruit Ninja Classic is 1.99. Let's use these values in this exercise.

1. Assign a value of 698516 to a variable named rating_count_totals.
   - This is the rating count total of Fruit Ninja Classic
2. What is the combined rating count total for Fruit Ninja Classic and Minecraft: Pocket Edition? Update the value of rating_count_totals by adding 522012 to its current value to find out.
   - You must use the += operator.


```
fruit_ninja_classic = 698516
minecraft_Pocket_rating_count_total = 522012
rating_count_totals = 698516
rating_count_totals += minecraft_Pocket_rating_count_total
print("rating count total = ", rating_count_totals)

```
Output
```
rating count total =  1220528
```

3. Assign a value of 1.99 to a variable named fruit_ninja_totals.


```
fruit_ninja_totals = 1.99
```

4. What is the cost of purchasing Fruit Ninja Classic for four people? Update the value of fruit_ninja_totals by multiplying its current value by 4 to find out.
   - You must use the *= operator.

Display rating_count_totals and fruit_ninja_totals using print().

1.7 Python Data Types

In this part, we'll learn about data types in Python: integers, floats, and string by focusing on their application in data analytics and AI. Throughout this section, we'll learn core Python programming concepts in the context of real data, which will help us understand how to handle various data types in practical scenarios whether it analyzing mobile app metrics or training machine learning models. We'll be using data from the table below, which provides some information about five mobile applications from the iOS store:

track_name	price	currency	rating_count_tot	user_rating
Facebook	0.0	USD	2974676	3.5
Instagram	0.0	USD	2161558	4.5
Clash of Clans	0.0	USD	2130805	4.5
Fruit Ninja Classic	1.99	USD	698516	4.5
Minecraft: Pocket Edition	6.99	USD	522012	4.5

In Python, we can make computations with integers (like 1) and decimal numbers (like 6.99):

```
print(6.99 * 1)
print(0.0 + 1.99)
```

```
Output
6.99
1.99
```

In mathematics, integers aren't the same as decimal numbers, and Python acknowledges this difference. We can use the type() function to see the type of a value and confirm that Python distinguishes between integers and decimal numbers:

```
print(type(0))

print(type(0.0))
```

```
Output
int
float
```

Notice that the integer 0 has the int type and the decimal number 0.0 has the float type. All integers have the int type, and all decimal numbers have the float type.

In computer programming, we classify values into different types — or data types. The type of value offers the computer the required information about how to process that value. Depending on the type, the computer will know how to store a value in memory, or what operations we can and can't perform on a value.

int and float values have different types, but we can mix the values together in arithmetical operations. So we're not limited, for instance, to adding an int value only to another int value — we can add an int value to a float value:

```
print(0 + 6.99)
print(1.99 * 2)
```

```
Output
6.99
3.98
```

1.7.1 Ex.

1. Create a variable to record personal app store purchases, and update the variable to reflect purchasing Fruit Ninja Classic.
   a. Initiate the variable personal_apps by assigning the integer value of 0.
   b. Update the value of personal_apps by adding the float 1.99 to its current value. You can use the += operator.
2. Record the app cost for Minecraft: Pocket Edition, and update the variable to reflect purchasing it for three friends.
   a. Assign the float 6.99 to a variable named gift_apps.
   b. Update the value of gift_apps by multiplying its current value by the integer 3.
   c. You can use the *= operator.
3. Display personal_apps and gift_apps using print().

1.8 Conversion Between Types

Sometimes, we want to change the type of a number. For example, we might want to change a decimal number (a float) into a whole number (an integer), or the other way around. Python gives us tools to do this. To change an integer into a float, we use the float() function:

```
print(float(0))
```

```
Output
0.0
```

If we want to change a float into an integer, we can use the int() function:

```
print(int(1.99))
```

```
Output
1
```

Notice that the int() function rounded 1.99 down to 1. The int() function will always round a float down, even if the decimal part is larger than 0.5.

```
print(int(6.99))
```

```
Output 6
```

If we want to round a number to the nearest whole number, we can use the round() function:

```
print(round(6.99))

print(round(1.99))

print(round(0.0))
```

```
Output
7
2
0
```

Notice that it's possible to combine functions. For instance, we encompassed a round() function within a print() function in each of the examples above. One thing to remember is that using the round() function doesn't actually change the value stored in a variable. To do that, we need to store the rounded value back into the variable:

```
minecraft_cost = 6.99
print(round(minecraft_cost))
print(minecraft_cost)

minecraft_cost = round(minecraft_cost)
print(minecraft_cost)
```

```
Output
7
6.99
7
```

1.8.1 Ex.

To complete this exercise, add your code on a new line below the provided code. Do not modify the existing code.

Provided code:

```
personal_apps = 0
personal_apps += 1.99
gift_apps = 6.99
gift_apps *= 3
print(personal_apps)
print(gift_apps)
# Update the values of personal_apps and gift_apps
# Write your code below this line
```

1. Round personal_apps using the round() function, and assign the rounded value back to personal_apps.
2. Convert gift_apps from a float to an integer using the int() function. Then, store the new integer value back in gift_apps.
3. Use the print() function to show the values stored in personal_apps and gift_apps.

1.9 Strings

So far, we've only worked with int and float values. But in data science and AI, numbers aren't the only type of data we work with. Let's refer once again to our dataset about five mobile applications from the iOS store:

track_name	price	currency	rating_count_tot	user_rating
Facebook	0.0	USD	2974676	3.5
Instagram	0.0	USD	2161558	4.5
Clash of Clans	0.0	USD	2130805	4.5
Fruit Ninja Classic	1.99	USD	698516	4.5
Minecraft: Pocket Edition	6.99	USD	522012	4.5

We can see that text (not numbers) represents the data in columns track_name and currency. In Python, we can create text by enclosing a sequence of characters within quotation marks (" "):

```
app_name = "Facebook"
currency = "USD"

print(app_name)
print(currency)
```

```
Output
Facebook
USD
```

Python syntax allows both double quotation marks (" ") and single quotation marks (' '). So, if we want to create the word "Facebook," we can use either "Facebook", or 'Facebook'.

```
fb_1 = "Facebook"
fb_2 = 'Facebook'

print(fb_1)
print(fb_2)
```

```
Output
Facebook
Facebook
```

In programming, we call sequences of characters like "Facebook", "USD", or "dasdaslkj" strings. In Python, a string is of the str type:

```
print(type('Facebook'))
```

```
Output
str
```

When we create strings, we can also use numbers, spaces, or other characters:

```
game = 'Clash of Clans'
short_description = 'Clash of Clans is free and has an average
rating of 4.5'

print(game)
print(short_description)
```

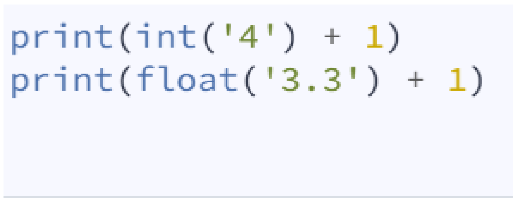
```
Output
Clash of Clans
Clash of Clans is free and has an average rating of 4.5
```

1.9.1 Ex.

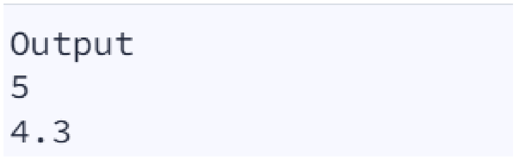
1. Assign the string Minecraft: Pocket Edition to a variable named app_name.
2. Assign the string 4.5 to a variable named average_rating. Don't mistake a string for a float.
3. Assign the string 522012 to a variable named total_ratings. Don't mistake a string for an integer.
4. Display the app_name variable using print().

1.10 String Conversion

If the strings contain characters that form a valid number (like '4', '3.3', '12', etc.), it's possible to convert them to integers or floats first, and then do the arithmetical operations. We can use the int() or float() function to convert a string of type str to a number of type int or float.

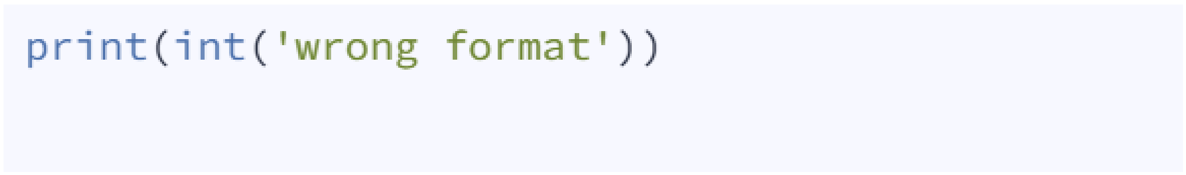


```
print(int('4') + 1)
print(float('3.3') + 1)
```

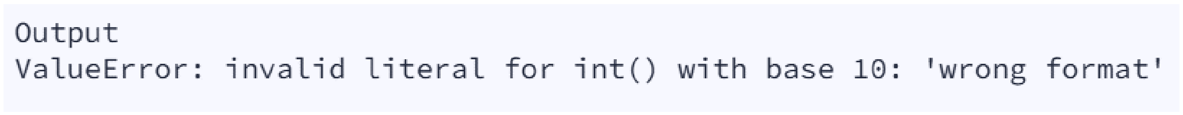


```
Output
5
4.3
```

But attempting to convert a str to an int won't work:

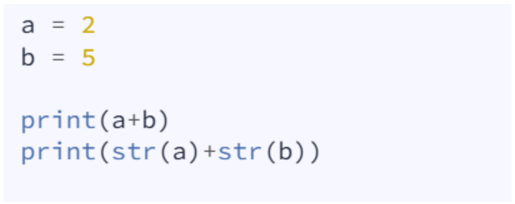


```
print(int('wrong format'))
```



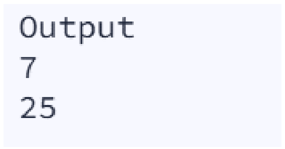
```
Output
ValueError: invalid literal for int() with base 10: 'wrong format'
```

Note that we can also convert an int or a float to a str using the str() function. In doing so, it will now have properties of a string as demonstrated below.



```
a = 2
b = 5

print(a+b)
print(str(a)+str(b))
```



```
Output
7
25
```

Notice that while the addition of the integers printed the sum of a and b, when converted into a string by the str() function it printed the concatenation of the strings instead.

1.10.1 Ex.

1. Assign the string Facebook's rating is to a variable named facebook.
2. Assign the float 3.5 to a variable named fb_rating.
3. Convert fb_rating from a float to a string using the str() function, and assign the converted value to a new variable named fb_rating_str.

Concatenate the strings stored in facebook and fb_rating_str to form the string Facebook's rating is 3.5.

   a. Assign the concatenated string to a variable named fb.
   b. You'll need to add a space character between Facebook's rating is and 3.5 to avoid ending up with the string Facebook's rating is3.5.

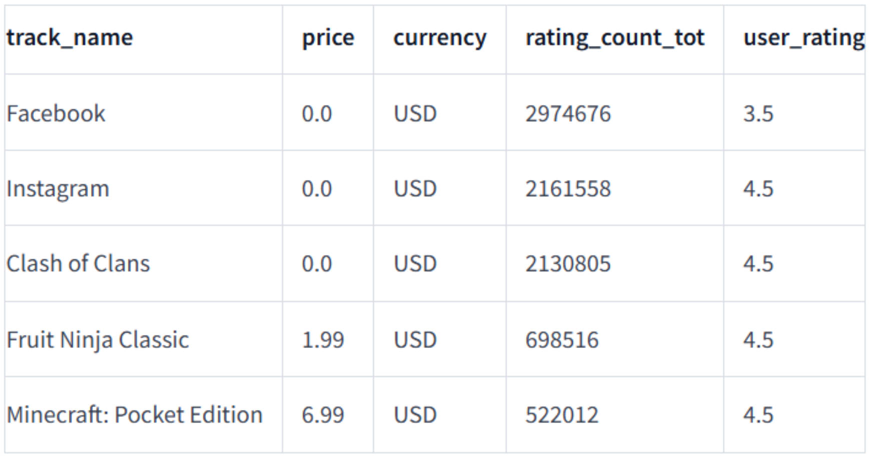
4. Display the fb variable using print().

1.11 Python Lists

In this section, we'll work with one of the most commonly used data structures in Python (lists). We'll also explore one of the most prevalent methods of

   a. organizing data
   b. arranging data by rows and columns into tables
   c. and how to model that in Python.

In the following table, each value is a data point:



track_name	price	currency	rating_count_tot	user_rating
Facebook	0.0	USD	2974676	3.5
Instagram	0.0	USD	2161558	4.5
Clash of Clans	0.0	USD	2130805	4.5
Fruit Ninja Classic	1.99	USD	698516	4.5
Minecraft: Pocket Edition	6.99	USD	522012	4.5

The first row has five data points:

- Facebook
- 0.0
- USD
- 2974676
- 3.5

A collection of data points makes up a dataset. Our table above is a collection of data points, so we call the entire table a dataset. We can see that our dataset has five rows and five columns.

When we work with datasets, we need to store them in the computer memory to be able to retrieve and manipulate the data points. We might think we could store each data point in a variable — for instance, this is how we might store the first row's data points:

```python
track_name_row_1 = 'Facebook'
price_row_1 = 0.0
currency_row_1 = 'USD'
rating_count_tot_row_1 = 2974676
user_rating_row_1 = 3.5
```

Above, we stored the following:

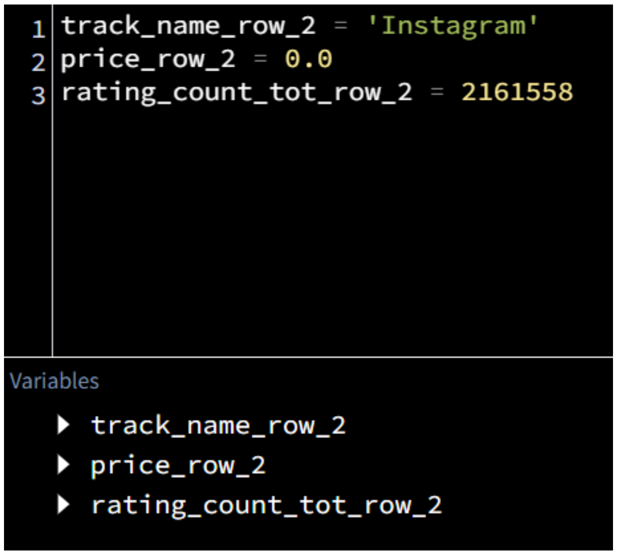
- The text "Facebook" as a string
- The price 0.0 as a float
- The text "USD" as a string
- The rating count 2,974,676 as an integer
- The user rating 3.5 as a float

Creating a variable for each data point in our dataset would be a cumbersome process. Fortunately, we can store data more efficiently using lists. We'll learn about that next. For now, let's practice storing row elements into variables.

1.11.1 Example

1. Store the track_name (app name) from the second row to a variable named track_name_row_2.
2. Store the price from the second row to a variable named price_row_2.
3. Store the rating_count_tot from the second row to a variable rating_count_tot_row_2.

Be sure to specify the correct data type for each variable. Use the table from the Learn section above to determine the data type to use for each variable.



```python
track_name_row_2 = 'Instagram'
price_row_2 = 0.0
rating_count_tot_row_2 = 2161558
```

1.12 Storing Rows as Lists

This is how we can create a list of data points for the first row:

```python
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]
print(row_1)
print(type(row_1))
```

```
Output
['Facebook', 0.0, 'USD', 2974676, 3.5]
```

To create the list above, we do the following:

- Type out a sequence of data points, and separate each with a comma: 'Facebook', 0.0, 'USD', 2974676, 3.5
- Surround the sequence with brackets: ['Facebook', 0.0, 'USD', 2974676, 3.5]

After we create the list, we store it in the computer's memory by assigning it to a variable named row_1.

To create a list of data points, we need to do the following:

- Separate the data points with a comma
- Surround the sequence of data points with brackets

Now let's practice creating lists.

1.12.1 Ex.

1. Store the second row ('Instagram', 0.0, 'USD', 2161558, 4.5) as a list in a variable named row_2.
2. Store the third row ('Clash of Clans', 0.0, 'USD', 2130805, 4.5) as a list in a variable named row_3.

1.13 List Length

A list can contain both mixed and identical data types. A list like [4, 5, 6] has identical data types (only integers), while the list ['Facebook', 0.0, 'USD', 2974676, 3.5] has mixed data types:

- Two strings ('Facebook', 'USD')
- Two floats (0.0, 3.5)
- One integer (2974676)

The ['Facebook', 0.0, 'USD', 2974676, 3.5] list has five data points. To find the length of a list, we can use the len() function:

```
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]
print(len(row_1))

list_1 = [1, 6, 0]
print(len(list_1))

list_2 = []
print(len(list_2))
```

```
Output
5
3
0
```

For small lists, we can just count the data points on our screens to find the length, but the len() function will prove very useful later when we work with lists containing thousands of elements. For now, let's practice assigning the length of a list to a variable.

1.13.1 Ex.

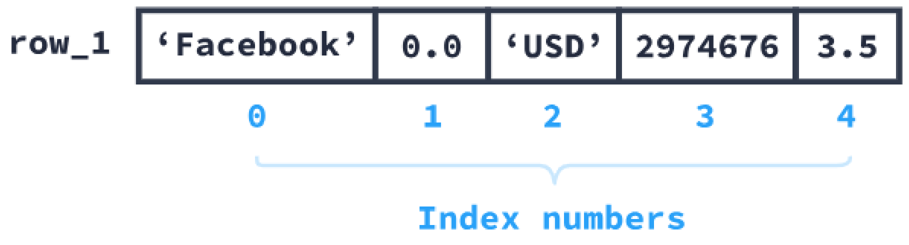
Given the lists row_2 and empty_row bellow.

```
row_2 = ['Instagram', 0.0, 'USD', 2161558, 4.5]
empty_row = []
```

1. Use the len() function to find the length of row_2, and store the result in a variable named row_2_length.
2. Use the len() function to find the length of empty_row, and store the result in a variable named empty_row_length.

1.14 List Indexing

Each element (data point) in a list has a specific number associated with it — this is an index number. The indexing always starts at 0, so the first element will have the index number 0, the second element will have the index number 1, and so on.



To quickly find the index of a list element, identify its position in the list, and then subtract

1. For example, the string 'USD' is the third element of the list (position number 3), so its index number must be 2 since 3−1=2

1.14.1 Ex.

Given the Lists below

```
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]
row_2 = ['Instagram', 0.0, 'USD', 2161558, 4.5]
row_3 = ['Clash of Clans', 0.0, 'USD', 2130805, 4.5]
```

1. Determine the index number of the track_name element 'Instagram' in row_2, and store the answer as an integer in a variable named track_name_index.
   - For example, the currency element 'USD' of row_1 is index number 2, so we'd write currency_index = 2.
2. Determine the index number of the user_rating element 4.5 in row_3, and store the answer as an integer in a variable named user_rating_index.

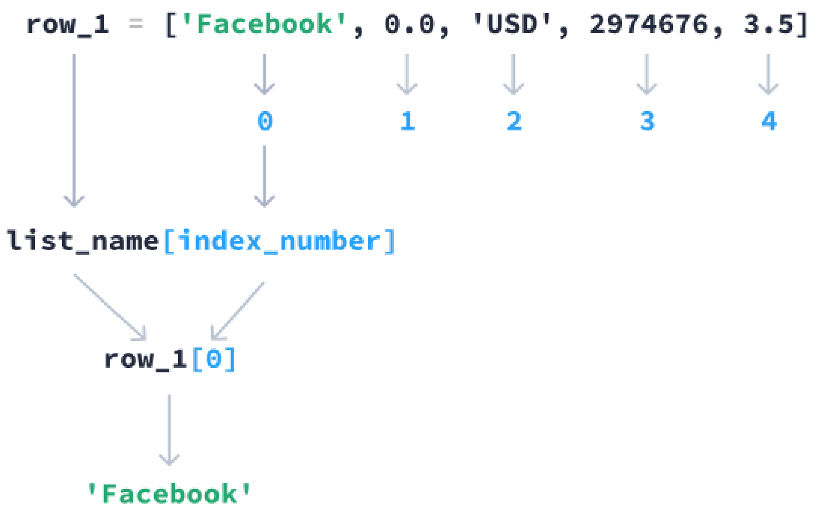
1.15 Retrieving Values from Lists

The index numbers help us retrieve individual elements from a list. Looking back at the list row_1 from the previous code example (shown below), we can retrieve the first element (the string 'Facebook') with the index number 0 by running the code print(row_1[0]).

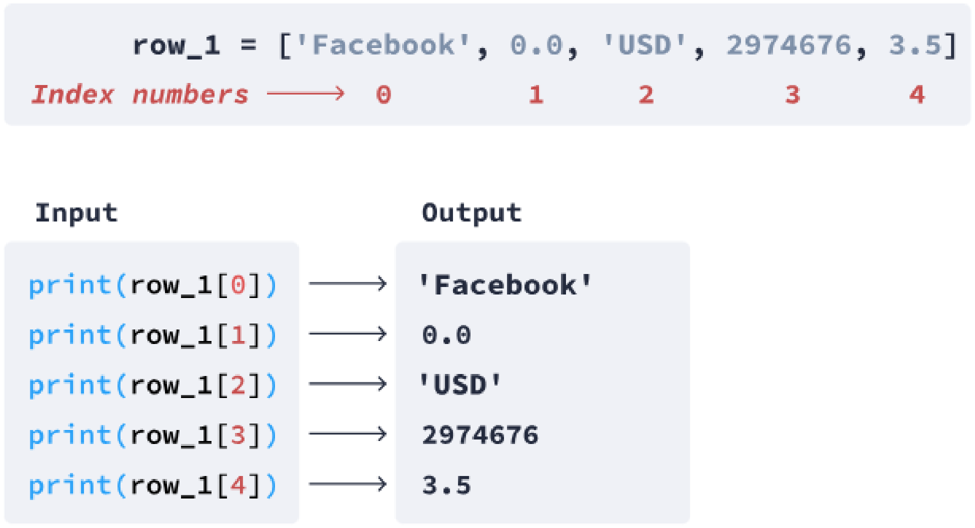
```
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]
print(row_1[0])
```

```
Output
Facebook
```

The syntax for retrieving individual list elements follows the model list_name[index_number]. For instance, the name of our list above is row_1, and the index number of the first element is 0 — following the list_name[index_number] model, we get row_1[0], where the index number 0 is in square brackets after the variable name row_1.



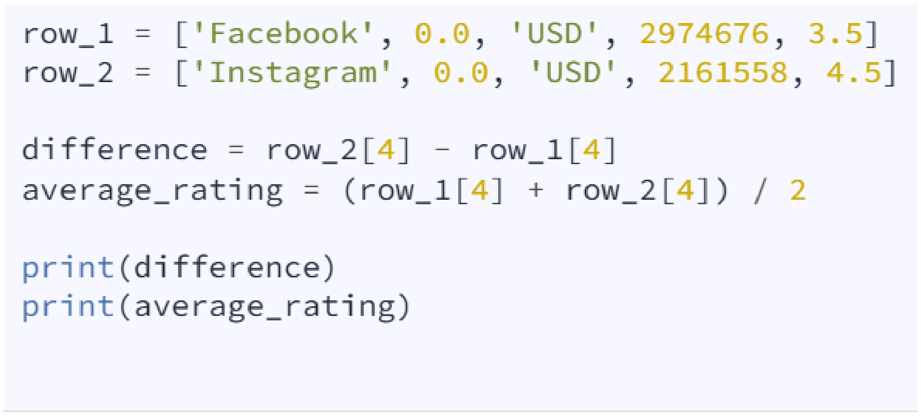
This is how we can retrieve each element in row_1:



```
      row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]
Index numbers ⟶ 0        1    2      3        4
```

```
Input                     Output
print(row_1[0])   ⟶   'Facebook'
print(row_1[1])   ⟶   0.0
print(row_1[2])   ⟶   'USD'
print(row_1[3])   ⟶   2974676
print(row_1[4])   ⟶   3.5
```

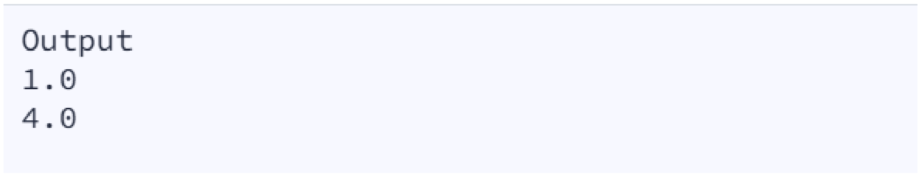
Retrieving list elements makes it easier to perform operations. For instance, we can select the ratings for Facebook and Instagram and find the average or the difference between the two:



```
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]
row_2 = ['Instagram', 0.0, 'USD', 2161558, 4.5]

difference = row_2[4] - row_1[4]
average_rating = (row_1[4] + row_2[4]) / 2

print(difference)
print(average_rating)
```



```
Output
1.0
4.0
```

1.15.1 Ex.

Give the lists for the first three rows bellow:

row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]

row_2 = ['Instagram', 0.0, 'USD', 2161558, 4.5]

row_3 = ['Clash of Clans', 0.0, 'USD', 2130805, 4.5]

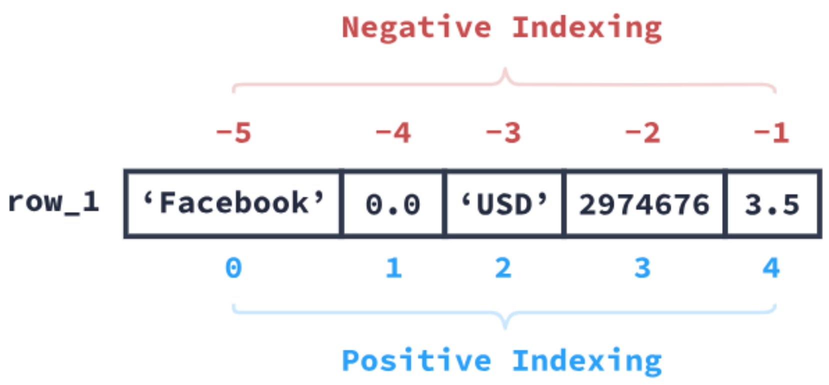
The fourth element in each list describes the number of ratings an app has received. Retrieve the fourth element from each list, and then find the average value of the retrieved numbers.

1. Assign the fourth element from the list row_1 to a variable named ratings_1. Don't forget that the indexing starts at 0.
2. Assign the fourth element from the list row_2 to a variable named ratings_2.
3. Assign the fourth element from the list row_3 to a variable named ratings_3.
4. Add the three numbers retrieved together and save the sum to a variable named total.
5. Divide the sum (now saved in the variable total) by 3 to get the average number of ratings for the first three rows. Assign the result to a variable named average.

1.16 Negative Indexing

In Python, we have two indexing systems for lists:

- Positive indexing: the first element has the index number 0, the second element has the index number 1, and so on.
- Negative indexing: the last element has the index number -1, the second to last element has the index number -2, and so on.



In practice, we almost always use positive indexing to retrieve list elements. Negative indexing is useful when we want to select the last element of a list — especially if the list is long, and we can't tell the length by counting.

```
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]

print(row_1[-1])
print(row_1[4])
```

```
Output
3.5
3.5
```

Notice that if we use an index number that is outside the range of the two indexing systems, we'll get an IndexError.

```
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]
print(row_1[6])
```

```
Output
IndexError: list index out of range
```

1.17 Retrieving Multiple List Elements

Sometimes we want to get more than one piece of information from a list. For example, if we have a list like ['Facebook', 0.0, 'USD', 2974676, 3.5], we might only want the name of the app and the details about its ratings (how many ratings it has and what the rating is). Here's how we can get those specific parts from the list:

```python
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]
app_name = row_1[0]
n_of_ratings = row_1[3]
rating = row_1[-1]
```

If we try to get this information for many apps, we'll end up with a lot of separate pieces of data, and our code can become long and confusing. A simpler way to handle this is to put the information we want into a new list.

```python
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]

fb_rating_data = [row_1[0], row_1[3], row_1[-1]]
print(fb_rating_data)
```

```
Output
['Facebook', 2974676, 3.5]
```

In the example above, we took three specific items from the list by using the code row_1[0], row_1[3], row_1[-1]. This code gets the first, fourth, and last items from the list. We then put these items inside square brackets to make a new list. Now, let's try doing something similar with the data from the following table:

track_name	price	currency	rating_count_tot	user_rating
Facebook	0.0	USD	2974676	3.5
Instagram	0.0	USD	2161558	4.5
Clash of Clans	0.0	USD	2130805	4.5
Fruit Ninja Classic	1.99	USD	698516	4.5
Minecraft: Pocket Edition	6.99	USD	522012	4.5

1.17.1 Ex.

1. For Facebook, Instagram, and Minecraft: Pocket Edition, create lists that contain the name of the app (track_name), the rating count (rating_count_tot), and the user rating (user_rating). Remember, the first item in a list is at index 0.
   - For Facebook, assign the list to a variable named fb_rating_data.
   - For Instagram, assign the list to a variable named insta_rating_data.
   - For Minecraft: Pocket Edition, assign the list to a variable named minecraft_rating_data.

2. Use the data in fb_rating_data, insta_rating_data, and minecraft_rating_data to find the average user rating for these three apps.
   - Add the three user ratings (user_rating) together and save the sum as total_rating.
   - Divide the total (total_rating) by 3 to get the average rating. Save it as average_rating.

1.18 Solved examples

Example 1:

Imagine you're tracking your daily water intake. Create a variable called `water_intake_liters` to represent the amount of water (in liters) you drank today. Assign it an initial value of `1.5`. Later, you drank another 0.7 liters. Update `water_intake_liters` accordingly and print the final value.

Solution:

```
python
Copy code
# Initial water intake in liters
water_intake_liters = 1.5

# Updating the intake after drinking more water
water_intake_liters += 0.7

# Display the final water intake
print("Total water intake:", water_intake_liters, "liters")
```

Example 2:

You're calculating the monthly budget for groceries. Create a variable called `groceries_budget` with a value of 200. Then, create another variable called `new_items` with a value of 45. Update `groceries_budget` by adding `new_items` and print the updated budget.

Solution:

```
python
Copy code
# Initial grocery budget
groceries_budget = 200

# Cost of new items
new_items = 45

# Update the budget after including new items
groceries_budget += new_items
```

```
# Display the updated grocery budget
print("Updated grocery budget:", groceries_budget)
```

Example 3:

You have a goal of saving 10% more each month. If your current savings are $1000, calculate your new savings goal by multiplying it by 1.1. Use a shorthand operator to update `savings` and print the result.

Solution:

```
python
Copy code
# Current savings
savings = 1000

# Increase savings by 10% using shorthand operator
savings *= 1.1

# Display the new savings goal
print("New savings goal:", savings)
```

Example 4:

You're organizing a marathon and need to track runners' timings in minutes. If you have a runner's time in seconds (`time_in_seconds = 4500`), convert it to minutes (as an integer) and print it.

Solution:

```
python
Copy code
# Time in seconds
time_in_seconds = 4500

# Convert time to minutes
time_in_minutes = time_in_seconds // 60

# Display the time in minutes
print("Runner's time:", time_in_minutes, "minutes")
```

Example 5:

You're designing a form that requires users to enter their height. Given the input as a string `height_cm = "175"`, convert it to an integer and calculate the height in meters by dividing by 100. Print the result.

Solution:

```
python
Copy code
# Height as a string
height_cm = "175"

# Convert height to integer and calculate in meters
height_m = int(height_cm) / 100
```

```
# Display height in meters
print("Height in meters:", height_m)
```

Example 6:

You're managing a list of weekly expenses in dollars: `[50, 30, 75, 20, 15]`. Calculate the total amount spent by summing up the list and print the total.

Solution:

```
python
Copy code
# List of weekly expenses
weekly_expenses = [50, 30, 75, 20, 15]

# Calculate the total expenses
total_expenses = sum(weekly_expenses)

# Display the total amount spent
print("Total amount spent:", total_expenses)
```

Example 7:

You're keeping track of daily temperatures in Celsius: `[25, 28, 30, 26, 27]`. Retrieve the temperature recorded on Wednesday (index 2) and print it.

Solution:

```
python
Copy code
# List of daily temperatures
temperatures = [25, 28, 30, 26, 27]

# Retrieve the temperature for Wednesday
wednesday_temp = temperatures[2]

# Display Wednesday's temperature
print("Wednesday's temperature:", wednesday_temp, "Celsius")
```

Example 8:

You recorded daily steps for a week: `[7000, 8500, 9200, 8800, 7900, 10500, 9500]`. Retrieve and print the steps you walked on the last day of the week using negative indexing.

Solution:

```
python
Copy code
# List of daily steps
daily_steps = [7000, 8500, 9200, 8800, 7900, 10500, 9500]

# Retrieve the steps on the last day
last_day_steps = daily_steps[-1]

# Display steps on the last day
```

```
print("Steps on the last day:", last_day_steps)
```

Example 9:

You're analyzing temperatures for a week `[22, 24, 19, 23, 20, 18, 21]`. Retrieve and print the temperatures for the middle of the week (from Wednesday to Friday) in a list.

Solution:

```
python
Copy code
# List of weekly temperatures
weekly_temperatures = [22, 24, 19, 23, 20, 18, 21]

# Retrieve temperatures for Wednesday to Friday
midweek_temperatures = weekly_temperatures[2:5]

# Display the temperatures for Wednesday to Friday
print("Midweek temperatures:", midweek_temperatures)
```

Example 10:

You have a list representing the weekly revenue in thousands of dollars: `[5.5, 6.2, 5.8, 6.7, 5.9]`. Increase each value by 10% to reflect an optimistic forecast and store the updated values in a new list called `forecasted_revenue`. Print the new list.

Solution:

```
python
Copy code
# List of weekly revenue
weekly_revenue = [5.5, 6.2, 5.8, 6.7, 5.9]

# Calculate forecasted revenue with a 10% increase
forecasted_revenue = [revenue * 1.1 for revenue in weekly_revenue]

# Display the forecasted revenue list
print("Forecasted revenue:", forecasted_revenue)
```

1.19 Exercises

Write python code to answer the following:

Question 1: You're managing a store and need to calculate the final price after a 15% discount on a product priced at $120. Create a variable `price` with an initial value of 120, apply the discount, and print the discounted price.

Question 2: Write a program to convert a temperature from Celsius to Fahrenheit. Assume the temperature is stored in a variable `celsius = 25`. Use the formula F=C×95+32.

Question 3: You've taken a loan of $1000 at an annual interest rate of 5% for 3 years. Write a program to calculate the simple interest using the formula Interest=Principal×Rate×Time. Print the interest.

Question 4: Write a program to check if a number `num = 27` is even or odd and print the result.

Question 5: You're creating a program to calculate the area of a circle with a given radius of 7. Use the formula Area=π×radius^2 and print the result.

Question 6: You're organizing a list of items in a shopping cart: `['apple', 'banana', 'milk', 'bread', 'cheese']`. Write a program to count the total number of items in the cart and print the count.

Question 7: You're tracking monthly sales for a quarter: `[3000, 4000, 5000]`. Retrieve the sales figure for February (index 1) and print it.

Question 8: You have a list of daily calorie intake: `[2000, 2200, 1800, 2100, 2500]`. You realize that the intake for the second day was actually 2300. Update the list and print the corrected list.

Question 9: You're tracking weekly expenses in dollars `[100, 150, 200, 120, 130]`. Calculate the total expenses for the week and print the result.

Question 10: You have a list of temperatures in degrees Celsius recorded over 5 days `[20, 22, 24, 19, 21]`. Calculate and print the average temperature for these days.