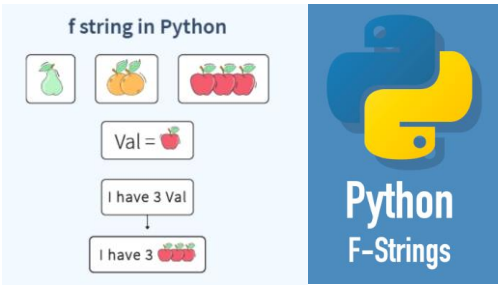
F string Formatting



Want a better way to insert a variable into an output string? The traditional way in Python 3 is done like this:

```
x="Mr. Walzl"
print('the person named',x,'is the best!')
```

but with f-string formatting with can get rid of the commas:

```
x= "Mr. Walzl"
print(f'the person named {x} is awesome')
```

f-string formatting isn't much different, but it is far more frequently used by experienced python programmers. Please notice the `f` after the print statement (it's an essential part of the syntax). You will notice later that f-string formatting is easier to read and write than multiple comma pairs.

Here are some more examples of how f-string formatting is done. For the first exercise run the following code in Replit to see what it does. Notice that we can **get rid of several (confusing) comma pairs** each time we use f-string formatting:

Exercise#1

Read the following does, **predict** what it will do, and then **run** in Replit to see what it does. Save your work as example#1

```
x='Laura'
y=25
z='smartest in the class.'
print(f'Out of {y} students {x} is {z}')
```

```
a=int(input('give me a number:'))
b=int(input('give me another number:'))
# here we an even evaluate an expression using f-string formatting
print(f'the sum of the two numbers you gave me is {a+b}')
```

Exercise#1 (continued)…**read** the following code, **predict** what you think it will do, then **run** the code in Replit and save as part of exercise#1

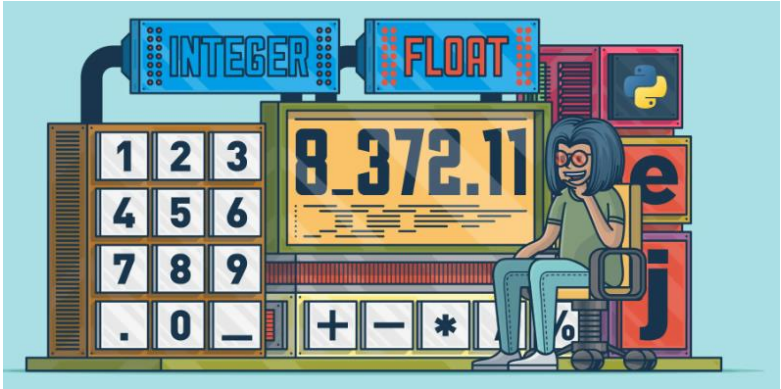
```
list1=['red','green','blue']

print(f'The first item in list1 is: {list1[0]}.  List1 has
{len(list1)} items in it.')
```

f-string Formatting

Decimal numbers

Exercise#2



Now let's use f-string formatting to **format decimal numbers** (floats).

Try the following code in Replit
Save them as **Exercise #2**:

```
number = float(input("Enter a decimal number with 4 decimal places: "))
print (f'{number}')
print (f'{number:.3f}')
print (f'{number:.2f}')
print (f'{number:.1f}')
```

f-string formatting also allows us to **format tables with clear columns**. Try the following code in Replit put it in **Exercise#2:**

```
header = ["Item","Price:"]
item1 =  ["Apples","$1.00"]
item2 =  ["Bananas","$0.75"]
item3 =  ["Cherries","$2.50"]

print(f"{header[0] : <20}{header[1] : <20}")
print(f"{item1[0] : <20}{item1[1] : <20}")
print(f"{item2[0] : <20}{item2[1] : <20}")
print(f"{item3[0] : <20}{item3[1] : <20}")
```

f-string formatting **tables with clear columns**.

Notice in the previous example that important part is the **specifier**: <20
This means: Create a column that is 20 spaces and then push text to the left < edge of the column.

```
print(f"{header[0] : <20}{header[1] : <20}")
print(f"{item1[0] : <20}{item1[1] : <20}")
print(f"{item2[0] : <20}{item2[1] : <20}")
print(f"{item3[0] : <20}{item3[1] : <20}")
```

Placing different symbols after a colon : in the { formatting } brackets allows you to tailor your formatting with a number of different options. Above we used the <20 symbol to create a column with 20 spaces and pushed the test to the left.

Please **watch the video** on the course page titled: **f-string format specifiers**.

Try the following and see what the **>** does. Include it in **Exercise#2**

```
s1 = 'Whistler Secondary School'
s2 = 'Where a helmet'
s3 = 'Enjoy'
s4 = 'WSS'

print(f'{s1 : >25}')
print(f'{s2 : >25}')
print(f'{s3 : >25}')
print(f'{s4 : >25}')
```

Watch the following video and then Try the following Exercises#3
https://www.youtube.com/watch?v=BbGGfTP1GZQ

Exercise#3

Write a Python program that prompts the user to enter two **decimal** numbers and then operation (+, -, *, or /) Then, **uses f-string formatting** to evaluate and display the result of the expression **neatly** to the user.

Here's the expected *flow* of the program:

a) Ask the user to input the first number (a float).
b) Ask the user to input the second number (a float).
c) Ask the user to input an operator (+, -, *, or /).
d) Display both numbers, the operation, and the result in **a neatly formatted print statement like the following example**

```
         23.456
+         9.340
---------------
         32.796
```

Helpful Hints:

- **Use f-string formatting** to **calculate and display the result of the expression**.
- You can use the `float()` function to convert user input to floating-point numbers.
- Use an if statement to check the operator entered by the user and perform the corresponding operation.

Exercise#4

The objective of this exercise is to practice using f-string formatting in Python to **create and display a simple table** from **two lists** provided by the user.

Write a Python program that creates a simple table of student names and their respective scores. The program should use f-string formatting to display the table neatly.

Here's the expected flow of the program:

a) Allow the **user** to create a **list** of student names (e.g., ["Alice", "Bob", "Charlie", "David"]).
b) Allow the **user** to create a corresponding **list** of scores for each student (e.g., [95, 87, 72, 88]).
c) Use a **for loop** and/or **zip( )** function (as shown below) to iterate through the lists then print student names and scores.
d) Use f-string formatting to display a table with **two columns**: one for the student names and one for their scores.
e) Format each row of the table using f-string formatting to align the columns neatly.

The following code *may* help you **display the student with the correct corresponding score** Your job is to see if you can use **f-string formatting** to place the output into a **neat table**. You can also insert the headings (Name: and Score:) if you wish.

```
num = [1, 2, 3]
color = ['red', 'while', 'black']
# iterates over 2 lists and executes
for (a, b) in zip(num, color):
 print (a, b)
```

for more on the **zip**() function check out:
https://blog.enterprisedna.co/python-zip-function-ultimate-guide-with-code-examples/

Sample intended output for Exercise#4:

```
Name:          Score:

Jeff           90
Carol          50
Tim            20
Denise         88
```