



# Fundamentals of Programming



## Printing Data in Python



Dr. Aamir Alaud Din



August 26, 2025

### Contents

1. Recap
2. Objectives
3. The Why Section
4. Printing Integers
5. Printing Floating Point Numbers
6. Printing in Scientific Notation
7. Printing Strings
8. Concatenation
9. Summary
10. Exercises

## 2. Objectives

After taking this lecture and studying, you should be able to:

- Print numeric data types in Python.
- Print numeric data types with control over data attributes.
- Concatenate the printing of different data types simultaneously.

## 3. The Why Section

- Consider an integer data type assigned to variable `x` and its printing as shown below.

```
x = 5  
print(x)
```

- The output of the above program is shown below.

```
5
```

- Is it possible to print `x = 5` with 5, 8, or any desired number of padded zeros?
- Now, consider the floating point data assigned to variable `y` and its printing as shown below.

```
y = 1.41  
print(y)
```

- The output of the above program is shown below.

```
1.41
```

- Can we print `y = 1.41` upto 4, 7, or 10 decimal places?
- Consider the following output of some program

```
C-C Bonds: 6  
C-C Bond Length: 1.40  
C-H Bonds: 6  
C-H Bond Length: 1.093  
Benzene has 12 bonds and average bond length is 1.2465
```

- The above output has `int`, `float`, and `str` data types.

- How can we print such a mixed data type simultaneously?
- The reason of studying this topic is not only to print data, but to learn printing with full control over required printing (individual and mixed data types).

## 4. Printing Integers

- Consider the following Python statement of assigning an integer to the variable `n_bonds`.
- Python's `print()` function is used to print data to standard output display device or LCD.
- We study different types of printing now.

### 4.1 Unformatted Printing

- The variable name is passed to the `print()` function as shown below.

```
n_bonds = 6
print(n_bonds)
```

- The output of the program is shown below.

```
6
```

### 4.2 Formatted Printing

- In formatted printing, we have control over printing.
- For formatted printing f-string printing is used.

#### 4.2.1 Unformatted Printing with F-string

- We print `n_bonds` with f-string as shown below.

```
n_bonds = 6
print(f"{n_bonds}")
```

- The output of the above program is as follows.

```
6
```

- In the above program `{}` is called the placeholder and it will become more clear in the subtopic of concatenation.

## 4.2.2 Informing Interpreter the Data Type

- We can also inform the interpreter that the data type is integer with letter `d`, called the format specifier, inside the place holder.
- Since variable is also written inside the place holder, so, variable name and format specifier must be separated by the colon operator like `{variable:format}`.
- The printing statement is given below.

```
print(f"{n_bonds:d}")
```

- The output will still be `6`.

## 4.2.3 Reserving Spaces with Right Alignment

- If we want to print `n_bonds` in 20 reserved spaces and aligned to right, we type `>20` just after the separator i.e., just after the colon operator.
- The symbol `>` tell interpreter that the printing is right aligned and the number `20` informs that the variable must be printed within 20 blank reserved spaces.
- This type of printing is shown below.

```
print(f"{n_bonds:>20d}")
```

- The output of the above program is shown below.

6

- The idea of reserved spaces and printing in these reserved spaces is shown in figure 1.



Figure 1. Right aligned printing of integer within 20 reserved spaces.

## 4.2.4 Reserved Spaces with Left Alignment

- The printing is exactly the same as right aligned printing with a minor change that the `>` symbol for right aligned printing is replaced with `<` symbol for left aligned printing.
- This printing is shown in the program below.

```
print(f"{n_bonds:<20d}")
```

- The output of this program is shown below.

```
6
```

- If no alignment symbol is used, the Python prints right aligned by default.

#### 4.2.6 Middle Aligned Printing in Reserved Blank Spaces

- For centred printing or middle aligned printing, the alignment symbol `^` is used.
- The program for such a printing is shown below.

```
print(f"{n_bonds:^20d}")
```

- The output of the above Python code is shown below.

```
6
```

- There are 9 blank spaces before 6 and 10 blank spaces after 6.
- The reason of one less blank space on left than on right will be clear when we study the magic fonts in this lecture.

#### Printing with Left Padded Zeros

- Suppose we want to print 12 left padded zeros before the number 6, we can do it with the following Python statement.

```
print(f"{n_bonds:013d}")
```

- The output will contain 12 left padded zeros with the integer 6 on the 13<sup>th</sup> place.

```
00000000000006
```

## 5. Printing Floating Point Numbers

- Suppose we want to print the number 1.431 which is assigned to variable `bond_length`.
- If we use the unformatted printing by passing `bond_length` to the `print()` function or we use f-string printing with the use of only variable name inside the place holder, the Python prints the number as it is.
- The format specifier for floating point numbers is `f`.

- If format specifier is used, Python prints the floating point number to 6 decimal places by default.
- The above discussion can be summarized in the following three lines of Python code.

```
bond_length = 1.431
print(bond_length)
print(f"{bond_length}")
print(f"{bond_length:f}")
```

- The output of the above three Python statement are shown below.

```
1.431
1.431
1.431000
```

## 5.1 Controlling Decimal and Reserved Spaces

- If `.x` is used just after the separator, the number will be printed to `x` decimal places.
- For example, the printing of number to two decimal places can be accomplished with the following Python statement.

```
print(f"{bond_length:.2f}")
```

- The output of the above Python statement is as follows.

```
1.43
```

## 5.2 Controlling Decimal Places and Alignment in Reserved Spaces

- The alignment symbols for alignment are exactly the same as discussed in section 4 of this topic.
- The number of reserved spaces are specified just after the alignment symbol, if either of the alignment symbol is used or just after colon operator if no alignment symbol is used.
- If no alignment symbol is used, Python aligns the printing to right side by default.
- Below we give examples of left, right, and middle aligned printing to 5 decimal places in 15 reserved spaces.

- We will use the value of  $\pi$  i.e., 3.141592653589793 assigned to variable `pi`.

```
pi = 3.141592653589793
print(f"{pi:<15.5f}")
print(f"{pi:>15.5f}")
print(f"{pi:^15.5f}")
print(f"{pi:15.5f}")
```

- The output of the above Python statements are given below.

```
3.14159
      3.14159
    3.14159
      3.14159
```

- The terminologies learnt so far are scattered.
- We use figures below which not only summarize the integers and floats printing but also summarizes the terminologies used.

```
pi = 3.14159
print(f' { pi : < 20 . 4 f } ' )
```

**Figure 2.** Printing of floating point number to four places of decimal and aligned left in 20 reserved spaces.

```
pi = 3.14159
print( f' { pi : < 20 . 4 f } ' )
```

print function

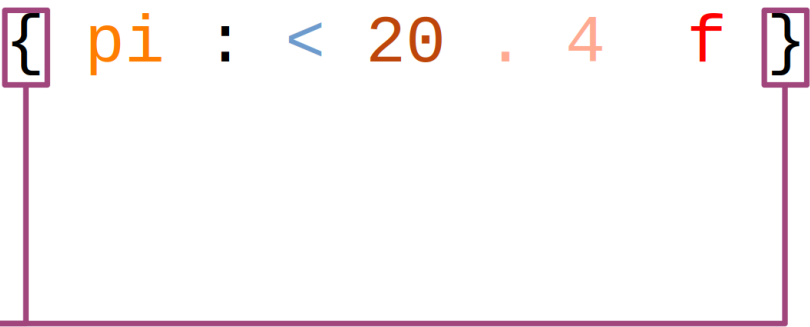
**Figure 3.** The `print()` function in printing floating point number.

```
pi = 3.14159
print( f' { pi : < 20 . 4 f } ' )
```

format string

**Figure 4.** Use of f-string in f-string printing.

```
pi = 3.14159
print( f' { } pi : < 20 . 4 f { } ' )
```




A purple line connects the two curly braces in the f-string to the label "place holder".

place holder

Figure 5. Use of place holder in f-string printing.

```
pi = 3.14159
print( f' { pi } : < 20 . 4 f { } ' )
```

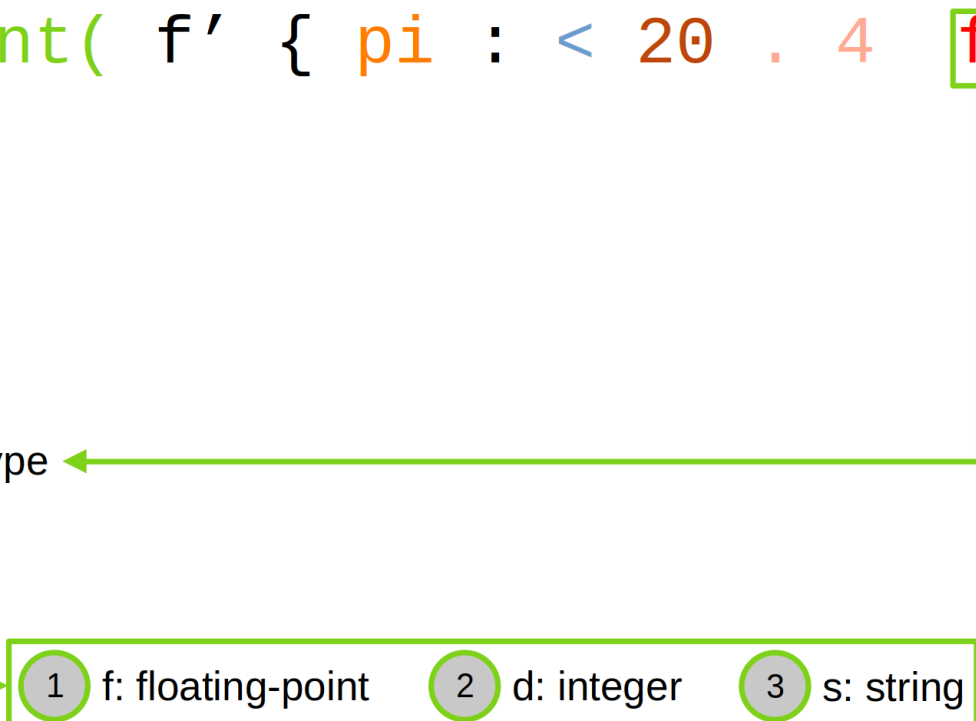


An orange line connects the variable name "pi" inside the curly brace to the label "variable name".

variable name

Figure 6. Use of variable names inside place holder of f-string.

```
pi = 3.14159
print( f' { pi : < 20 . 4 f } ' )
```



A green line connects the 'f' in the format specifier to the label "data type". Another green line connects the "data type" label to a legend box containing three items: "1 f: floating-point", "2 d: integer", and "3 s: string".

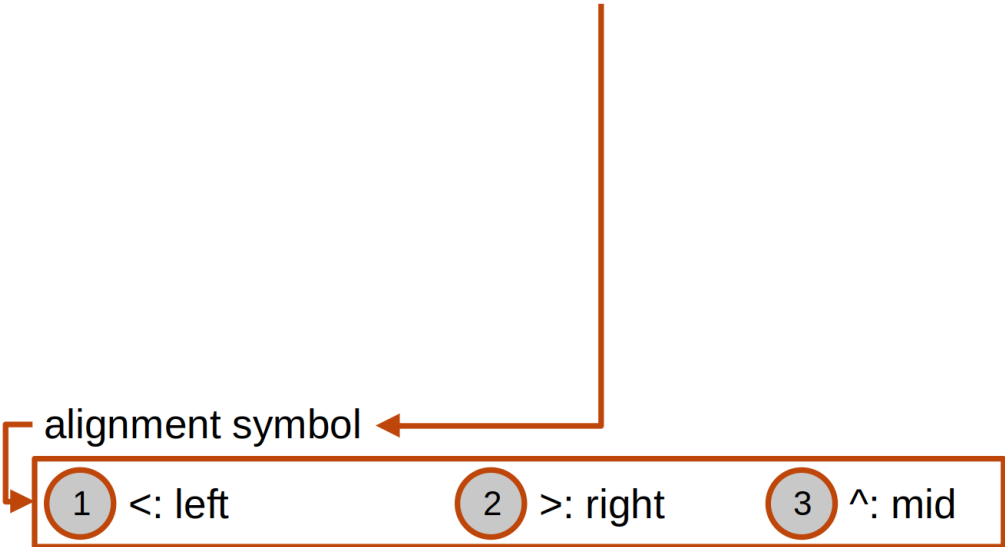
data type

1 f: floating-point   2 d: integer   3 s: string



Figure 7. Use of format specifier inside place holder in f-string printing.

```
pi = 3.14159
print( f' { pi : < 20 . 4 f } ' )
```

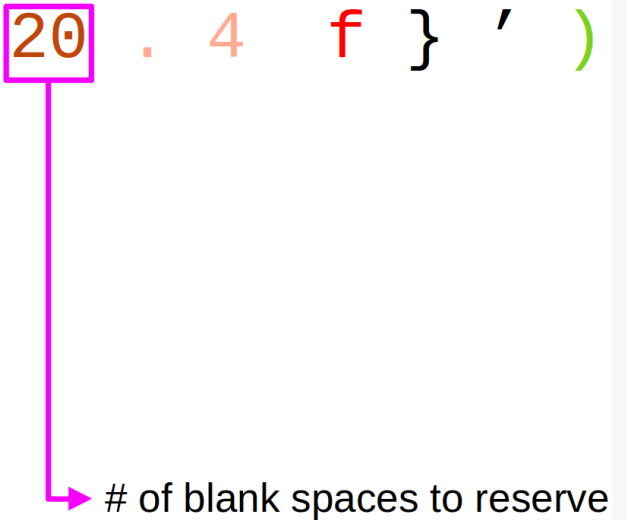


The diagram illustrates the format specifier `<` in the f-string `f' { pi : < 20 . 4 f } '`. An arrow points from the `<` symbol to a box containing three alignment options:

- 1 <: left
- 2 >: right
- 3 ^: mid

Figure 8. Use of alignment symbol inside place holder in f-string printing.

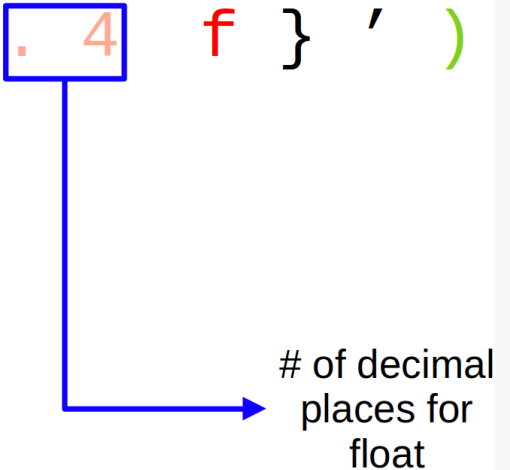
```
pi = 3.14159
print( f' { pi : < 20 . 4 f } ' )
```



The diagram illustrates the number of blank spaces reserved in the format specifier `20` in the f-string `f' { pi : < 20 . 4 f } '`. A purple arrow points from the `20` to the text `# of blank spaces to reserve`.

Figure 9. Reserving number of blank spaces inside place holder in f-string printing for printing of variable.

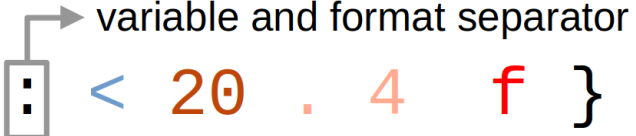
```
pi = 3.14159
print( f' { pi : < 20 . 4 f } ' )
```



# of decimal places for float

Figure 10. Fixation of the number of decimal of a floating point number inside place holder in f-string printing.

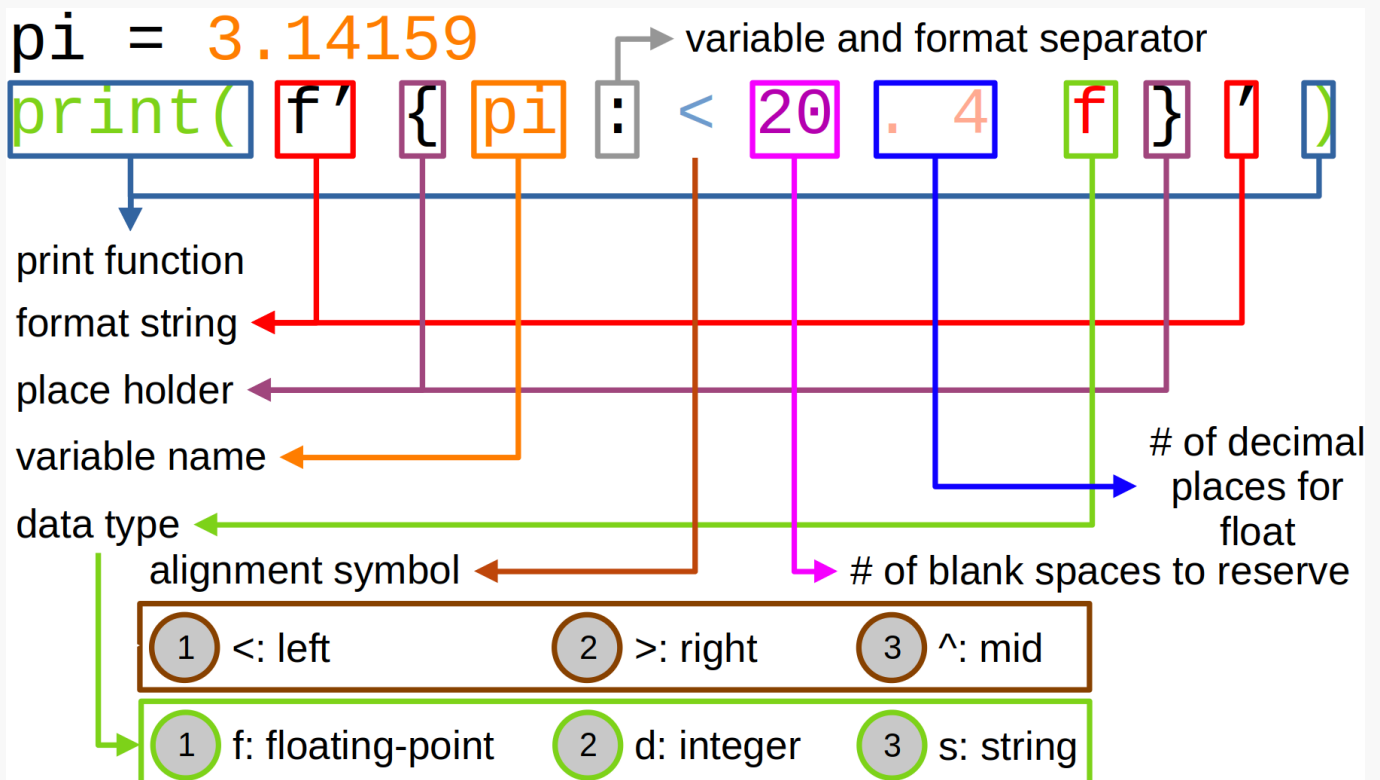
```
pi = 3.14159
print( f' { pi : < 20 . 4 f } ' )
```



variable and format separator

Figure 11. Use of colon operator for variable and format separation inside place holder in f-string printing.

```
pi = 3.14159
print( f' { pi : < 20 . 4 f } ' )
```



print function

format string

place holder

variable name

data type

alignment symbol

# of decimal places for float

# of blank spaces to reserve

1 <: left	2 >: right	3 ^: mid
1 f: floating-point	2 d: integer	3 s: string

Figure 12. Complete summary of printing with f-string.

[illegible][illegible][illegible]

## 6. Printing in Scientific Notation

- ```
na = 6022140760000000000000000000000000  
print(f"{na:18.3E}")
```

- The output of this program is as shown below.

```
6.022E+23
```

- We notice that reading large number like Avogadro's number in computer programs may create confusion because thousand separation commas are not allowed within numeric data in Python.
- However, Python allows the use of underscores in place of commas for ease of program readability in integers and floats.
- So, the variable assignment `na = 602_214_076_000_000_000_000_000` is exactly the same as `na = 602214076000000000000000` which is without underscores.
- The print output of both assignments is always without underscores as shown below.

```
602214076000000000000000
```

## 7. Printing Strings

- String can be passed either directly to the `print()` function or the variable containing the string can be passed to the `print()` function.
- The following print statements show printing of string by passing string directly and passing the variable containing string to the `print()` function.

```
print("Computer programming is logical.")  
prog_state = "Computer programming is logical."  
print(f"{prog_state}")
```

- The output of the above statements is the same which is shown below.

```
Computer programming is logical.  
Computer programming is logical.
```

- The format specifier for printing of strings is `s`.
- The printing of the above variable in 50 reserved spaces with different alignments is shown in following Python print statements and are self explanatory.

```
print(f"{prog_state:<50s}")
print(f"{prog_state:>50s}")
print(f"{prog_state:^50s}")
```

- The output of these lines is shown below.

Computer programming is logical.

Computer programming is logical.

Computer programming is logical.

- Python overrides the user input number of blank spaces if length of string is longer than the number of reserved spaces provided by the user.
- For example, if we try to print `prog_state` in 10 reserved spaces, Python will automatically reserve 32 spaces for the string and will print it.
- More controls of printing strings will be included when we will study strings in detail.

## 8. Concatenation

- So far, we discussed the printing of integers, floating point numbers, and strings individually.
- But, how to print if we have a mixture of data types?

### Definition: Concatenation

The art of printing mixed data types is called concatenation.

- The place holders inside the f-string decide the location where the variable is to be printed.
- Suppose, we have two variables which are `cc_bonds` and `cc_bond_length` to which the `int` and `float` data types are assigned as shown below.

```
cc_bonds = 6
cc_bond_length = 0.14
```

- Various print statements can be used to print `There are 6 C-C bonds and bond length of each C-C bond is 0.1400 nm.`
- We show few varieties to do the job.

```

cc_bonds = 6
cc_bond_length = 0.14
print(f"There are {cc_bonds:d} C-C bonds and bond length of each C-C
bond is {cc_bond_length:.4f} nm.")
print(f"There are{cc_bonds:2d} C-C bonds and bond length of each C-C
bond is{cc_bond_length:7.4f} nm.")
print(f"There are{cc_bonds:^3d}C-C bonds and bond length of each C-C
bond is{cc_bond_length:^8.4f}nm.")
str1 = "There are"
str2 = "C-C bonds and bond length of each C-C bond is"
str3 = "nm."
print(f"{str1:<10s}{cc_bonds:<2d}{str2:<46s}{cc_bond_length:<7.4f}
{str3:<4s}")

```

- Suppose, we want to print the following statments.

```

There are 6 C-C bonds.
The bond lenght of each C-C bond is 0.1400 nm.

```

- The following two different statement do the job.

```

print(f"There are {cc_bonds:d} C-C bonds.")
print(f"The bond length of each C-C bond is {cc_bond_length:.4f} nm.")

```

- Another version which does the same job is shown below.

```

print(f"There are {cc_bonds:d} C-C bonds.\nThe bond length of each C-C
bond is {cc_bond_length:.4f} nm.")

```

- The characters `\n` is called the escape character and is called the line feed i.e., new line.
- Similarly, there are other escape characters which do different jobs.

## 9. Summary

- In unformatted printing, variable is directly passed to the `print()` function as input argument.
- In formatted printing f-string printing is used.
- In f-string, the place holder is the location of printing data.
- In f-string printing, the data types inside place holder are represented by the following characters.

1. The character `d` is used to represent integers.
  2. The character `f` is used to represent floating point numbers.
  3. The character `e` is used to represent data in scientific notation.
  4. The character `s` is used to represent strings.
- The printing alignment in reserved spaces is controlled with following three symbols.
    1. The character `<` is used for left aligned printing.
    2. The character `>` is used for right aligned printing.
    3. The character `^` is used for centered printing.
  - Printing of mixed data types requires concatenation with knowledge of escape characters.

## 10. Exercises

### Exercise 1

Print the following variables using f-string.

| Variable              | Data Type | Reserved Spaces | Alignmen<br>t | Notation   | Decimal Places |
|-----------------------|-----------|-----------------|---------------|------------|----------------|
| c=299792458           | int       | 20              | Right         | N/A        | N/A            |
| name="Aamir AlaudDin" | str       | 22              | Middle        | N/A        | N/A            |
| c=299792458           | int       | 20              | Right         | Decimal    | 2              |
| na=6.022E+23          | float     | 10              | Left          | Scientific | 4              |
| me=9.109e-31          | float     | 15              | Middle        | Scientific | 2              |
| me=9.109E-31          | float     | 40              | Left          | Decimal    | 34             |
| R=0.08206             | float     | 8               | Left          | Scientific | 2              |
| kb=1.380649E-23       | float     | 20              | Right         | Scientific | 4              |
| kb=1.380649E-23       | float     | 30              | Middle        | Scientific | 5              |
| kb=1.380649E-23       | float     | 29              | Left          | Decimal    | 29             |



## Exercise 2

Python escape characters are used to format f-string and a list of escape characters with examples is given [here](#). You are required to use each escape sequence in your own preferred string like the one given below.

```
test="My name is Aamir Alaud Din. I teach Fundamentals of Programming  
at SCEE (IESE)."
```

## Exercise 3

Consider the following statement

```
milk = "We are 6 members and we consume 2.25 L milk daily."
```

Assign variables and use them to print the above string as shown below.

```
str1 = "We are"  
members = 6  
str1 = "members and we consume"  
usage = 2.25  
str3 = "L milk daily."  
print(f"{str1:<7s}{members:d}{str2:^24s}{usage:<5.2f}{str3}")
```

Print the following statements using f-string just like the example given above.

- In a sodium atom, there are 11 electrons and the mass of one electron is  $9.109 \times 10^{-31} \text{ kg}$ .
- The speed of light in vacuum is approximately  $3.00 \times 10^8 \text{ m/s}$ , a constant value in physics.
- A single water molecule ( $\text{H}_2\text{O}$ ) contains 2 hydrogen atoms and 1 oxygen atom. Atomic masses of hydrogen and oxygen are 1.008 and 15.999 amu.
- The universal gas constant R equals  $8.314 \text{ J}/(\text{mol} \cdot \text{K})$ .
- The Avogadro number is  $6.022 \times 10^{23} \text{ mol}^{-1}$ , representing the number of entities in 1 mole.
- The freezing point of water is  $0^\circ\text{C}$  or  $273.15 \text{ K}$  under standard atmospheric pressure.
- The charge of a proton is  $+1.602 \times 10^{-19} \text{ C}$ , while the charge of an electron is negative of this value.

- h. The gravitational constant  $G$  is  $6.674 \times 10^{-11} N \cdot m^2/kg^2$ .
- i. The Planck constant  $h$  is  $6.626 \times 10^{-34} J \cdot s$ , used in quantum mechanics.
- j. Carbon-12 has an atomic mass of exactly  $12 u$ , by definition of the unified atomic mass unit and that of hydrogen is  $1.008 u$ .

**Note:**

- There must not be more than one space between any two adjacent words and numbers.
- If your output is not correct in the first go, first understand the problem and then modify the print statement accordingly so that your desired output is correct in the second go.
- Understand all the exercises logically, because, similar exercises may be part of quizzes, MSE, and ESE.
- Deadline is same day class in next week.