Lecture – Strings

- Python string formatting.

Python string formatting

- The following example formats a string using two variables and summarizes three string formatting options in Python.

```
name = 'Peter'
age = 23

print('%s is %d years old' % (name, age))       # method 1
print('{} is {} years old'.format(name, age))   # method 2
print(f'{name} is {age} years old')              # method 3
```

- Each method gives the same output:

```
     Peter is 23 years old
```

Python string formatting

- Method 1 – oldest option. Uses the % operator and types such as %s and %d.

```
    print('%s is %d years old' % (name, age))
```

- Method 2 – uses the format() function introduced in Python 3.0 to provide advance formatting options.

```
    print('{} is {} years old'.format(name, age))
```

- Method 3 – **newest option: python f-strings**.

```
    print(f'{name} is {age} years old')
```

Python f-string

- Available since Python 3.6.
- The string has the f (or F) prefix and uses {} to evaluate variables.
- Provide a **faster** and more **concise** way of formatting strings in Python.
- f-string is short for **formatted string literals**.

```
print(f'{name} is {age} years old')
```

Python f-string

- **Expressions** - We can put expressions between the {} brackets. These will be evaluated at the program runtime.

```
bags = 3
apples_in_bag = 12
print(f'Total of {bags * apples_in_bag} apples')
```

- Output:
```
Total of 36 apples
```

- **Methods** - We can call methods in f-strings.

```
print(f'My name is { name.upper() }')
```

- Output:
```
My name is JOHN DOE
```

f-string format specifiers

- Format specifiers are specified after the **colon character**.
- **Floating point values** have the f suffix. We can also specify the **precision**: the number of decimal places. The precision value goes right after the dot character.

```
    val = 12.335336
    print(f'{val:.2f}')
    print(f'{val:.5f}')
```

- Prints a formatted floating point value of 2 and 5 decimal places:

```
    12.34
    12.33534
```

f-string format specifiers

- Format specifiers are specified after the **colon character**.
- **width specifier** sets the width of the value. The value may be filled with spaces or other characters if the value is shorter than the specified width.

```
for x in range(1, 6):
    print(f'{x:02} {x*x:3} {x*x*x:4}')
```

- The example prints three columns. Each of the columns has a predefined width. The first column adds a leading 0 is the value is shorter than the specified width (the other columns use spaces). Output:

```
01   1    1
02   4    8
03   9   27
04  16   64
05  25  125
```

f-string format specifiers

- Format specifiers are specified after the **colon character**.
- By default the strings are left justified. Use > character to **justify to the right**.

```
s1 = 'a'
s2 = 'ab'
print(f'{s1:>10}')       # default - print(f'{s1:10}')
print(f'{s2:>10}')      # default - print(f'{s2:10}')
```

- Sets the width of output to **10 characters with values right justified**. Output:

```
         a
        ab
```

- Default output (without **>**) will left justify:

```
a
ab
```

Lecture Summary

- Python string formatting.

Curly brackets { } marks a replacement field

- Format specifiers are specified after the **colon character**.

```
    print(f'{x:02} {x*x:3} {x*x*x:4}’)
```

- To add the number of decimal palaces add a dot and precision value:

```
    print(f'{val:.2f}’)
```

- By default the strings are left justified. Use > character to **justify to the right**.

```
    print(f'{s1:>10}')
```