Tasks – Literals and Data Types

Practice Python literals: strings, integers, floats, octal, hexadecimal, and booleans. Each section has 2–3 short exercises. Create the file, write the code, run it, and check the output.

Run scripts with: `python3 script_name.py`

---

1. Strings

Task 1.1 – Single and double quotes (`strings_quotes.py`)

- Create `strings_quotes.py`.
- In one script: print one string using double quotes and one using single quotes (e.g. `"Hello"` and `'World'`).
- Run the script.

Expected output (example):

```
Hello
World
```

Task 1.2 – Escape sequences (`strings_escapes.py`)

- Create `strings_escapes.py`.
- Print a string that contains a newline (`\n`) so the output is two lines.
- Print a string that contains a tab (`\t`) between two words.

Expected output (example):

```
Line one
Line two
```

```
word    other
```

Task 1.3 – Multi-line string (`strings_multiline.py`)

- Create `strings_multiline.py`.
- Use a triple-quoted string (e.g. `"""..."""` or `'''...'''`) that spans two or three lines in your code.
- Print it. The output should show the same line breaks.

Expected output (example):

```
First line
Second line
Third line
```

---

2. Integers

Task 2.1 – Basic integers (`ints_basic.py`)

- Create `ints_basic.py`.
- Print a positive integer, a negative integer, and zero (each with its own `print()`).

Expected output:

```
42
-7
0
```

Task 2.2 – Large integers with underscores (`ints_large.py`)

- Create `ints_large.py`.
- In Python you can write large numbers with underscores (e.g. `1_000_000`). Assign such a number to a variable and print it.
- Print the result of a simple calculation (e.g. `1_000 + 500`).

Expected output (example):

```
1000000
1500
```

Task 2.3 – Integer arithmetic (`ints_arithmetic.py`)

- Create `ints_arithmetic.py`.
- Print the result of: `10 + 3`, `10 - 3`, `10 * 3`, and `10 // 3` (integer division). Use four `print()` calls.

Expected output:

```
13
7
30
3
```

---

3. Floats

Task 3.1 – Float literals (`floats_basic.py`)

- Create `floats_basic.py`.
- Print at least three float literals (e.g. `3.14`, `-0.5`, `2.0`).

Expected output (example):

```
3.14
-0.5
2.0
```

Task 3.2 – Scientific notation (`floats_scientific.py`)

- Create `floats_scientific.py`.

- Print a number using scientific notation (e.g. `1e3` or `1e-2`).
- Print another (e.g. `2.5e2`). Run and see the displayed value.

Expected output (example):

```
1000.0
250.0
```

Task 3.3 – Division and mixed types (`floats_mixed.py`)

- Create `floats_mixed.py`.
- Print the result of `10 / 4` (regular division – gives a float).
- Print the result of `10 // 4` (integer division – gives an int). Show that one is float and one is int by printing both.

Expected output:

```
2.5
2
```

---

4. Octal

Task 4.1 – Octal literals (`octal_basic.py`)

- Create `octal_basic.py`.
- In Python, octal literals start with `0o` (zero and the letter o). Assign `0o10` to a variable and print it.
- Print `0o755` (common in file permissions). The output should be the decimal value.

Expected output:

```
8
493
```

Task 4.2 – Convert decimal to octal (`octal_convert.py`)

- Create `octal_convert.py`.
- Use the built-in `oct()` function. Pass it the integer `255` and print the result.
- Print `oct(8)`.

Expected output:

```
0o377
0o10
```

Task 4.3 – Same value in octal and decimal (`octal_decimal.py`)

- Create `octal_decimal.py`.
- Assign the value `0o12` to a variable. Print that variable (displays in decimal).
- Print the same variable inside `oct()` to show it in octal form again.

Expected output:

```
10
0o12
```

---

5. Hexadecimal

Task 5.1 – Hex literals (`hex_basic.py`)

- Create `hex_basic.py`.
- Hex literals start with `0x`. Print the value of `0xFF`.
- Print the value of `0x1A`.

Expected output:

```
255
26
```

Task 5.2 – Convert to hex (`hex_convert.py`)

- Create `hex_convert.py`.
- Use `hex(255)` and print the result.
- Use `hex(16)` and print the result.

Expected output:

```
0xff
0x10
```

Task 5.3 – Hex “color” value (`hex_color.py`)

- Create `hex_color.py`.
- Pick a small hex number that could look like a color component (e.g. `0xFF` for 255, `0x80` for 128). Assign it to a variable, print the variable (decimal), then print `hex(variable)` to show the hex form.

Expected output (example):

```
255
0xff
```

---

6. Booleans

Task 6.1 – True and False (`bools_basic.py`)

- Create `bools_basic.py`.
- Print the literal `True`.
- Print the literal `False`.

Expected output:

```
True
False
```

Task 6.2 – Comparisons (`bools_comparison.py`)

- Create `bools_comparison.py`.
- Print the result of `3 > 2`.
- Print the result of `1 == 0`.
- Print the result of `5 <= 5`.

Expected output:

```
True
False
True
```

Task 6.3 – bool() conversion (`bools_convert.py`)

- Create `bools_convert.py`.
- Print `bool(0)` and `bool(1)`.
- Print `bool("")` and `bool("hello")`. See which are truthy and which are falsy.

Expected output:

```
False
True
False
True
```

Done

You've used: string literals (quotes, escapes, triple-quoted), integer and float literals, scientific notation, octal (`0o`, `oct()`), hexadecimal (`0x`, `hex()`), and booleans (`True`/`False`, comparisons, `bool()`).