# Variables, Type Conversion, Operators & Methods

## Session Outline:

- Assigning & Overwriting Variables; Naming Convention for Variables.
- Type Conversion.- Operators: Arithmetic & Logical/Comparison.
- Numeric & String (Text) Functions.- String Indexing & Slicing.
- String Methods.
- F String.

## 1. Assigning & Overwriting Variables:

**Variables:** Containers for storing data values.

**Assigning Variables:** Use the = operator.

**Overwriting Variables:** Reassign a new value to an existing variable.

**Naming Conventions:**

- Use descriptive names (e.g., user_age instead of a).
- Start with a letter or underscore (_).
- Use snake_case (e.g., user_name).

```
# Example: Assigning and Overwriting Variables
name = "Alice"  # Assigning
print("Name:", name)

name = "Bob"  # Overwriting
print("Updated Name:", name)

# Naming Convention Example
user_age = 25  # Descriptive and snake_case
print("User Age:", user_age)
```

```
Name: Alice
Updated Name: Bob
User Age: 25
```

**Real-life Use Case:**

Storing user input or configuration values.

**DELETING VARIABLES:**

The **del** keyword will permanently remove variables and other objects.

```
price = 3
del price

print(price)
```

Variables have some basic **naming rules**

**Variable names can:**

- Contain letters (case sensitive!)
- Contain numbers
- Contain underscores
- Begin with a letter or underscore

**Variable names cannot:**

- Begin with a number
- Contain spaces or other special characters (', &, ^, -, etc.)
- Be reserved Python keywords like **del** or **list**

**PRO TIP:** "Snake case" is the recommended naming style for Python variables, which is all lowercase with words separated by underscores (first_second_third, new_price, etc.)

**Valid** variable names:

- price_list_2019
- _price_list_2019
- PRICE_LIST_2019
- pl2019

**Invalid** variable names

- 2019_price_list *(starts with a number)*
- price_list_2019 *(has special characters)*
- 2019 price list *(has spaces)*
- list *(reserved Python keyword)*

**TRACKING VARIABLES:**

Use **"%who"** and **"%whos"** to track the variables you've created.

```
price = 10
product = 'Super Snowboard'
Date = '10-Jan-2021'
dimensions = [160, 25, 2]
```

```
# %who returns variable names
```

```
%who
```

```
# %whos returns variable names, types, and information on the data contained
```

```
%whos
```

**Magic commands** that start with **"%"** only work in the iPython environments, which applies to Jupyter and Colab.

## 2. Type Conversion

**Type Conversion:** Converting one data type to another.

**Implicit Conversion:** Automatically done by Python (e.g., int to float).

**Explicit Conversion:** Done using functions like int(), float(), str(), etc.

```
# Example: Type Conversion
# Implicit Conversion
num1 = 10  # int
num2 = 5.5  # float
result = num1 + num2  # Automatically converts to float
print("Result:", result, "Type:", type(result))

# Explicit Conversion
age = "25"  # String
age_int = int(age)  # Convert to integer
print("Age as Integer:", age_int, "Type:", type(age_int))
```

```
Result: 15.5 Type: <class 'float'>
Age as Integer: 25 Type: <class 'int'>
```

**Real-life Use Case:**

Converting user input (always a string) to numbers for calculations.

```
type(1), 3, 5, 7, 9))
```

```
type(1), 3, 5, 7, 9))
```

Like typo() if you are getting a TypeError or unexpected behavior when passing data through a function to make sure the data type is correct.

It's not uncommon for values to be stored incorrectly.

```
#Converting data into an integer data type

print(type("123"))  #initially, '123' refers to string as it is under quotation

int("123")  #Using int converts the text string of '123' into an integer data type

#to check the type after conversion

print(type(int("123")))
```

```
print(type(int((1,2,3,5))))  #This will throw a TypeError as let attempts to convert the list into an integer data type
```

```
type(set((1,2,3,5)))
```

```
set((1,2,3,5))
```

```
len((1,2,3,5))
```

```
len(set((1,2,3,5)))
```

## 3. Operators:

**Arithmetic Operators:**

Used for mathematical operations: +, -, *, // (floor division), % (modulus), ** (exponentiation).

```
# Example: Arithmetic Operators

a = 10
b = 3
print("Addition:", a + b)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", a / b)
print("Floor Division:", a // b)  # Rounds down to the nearest integer
print("Modulus:", a % b)  # Remainder after division
print("Exponentiation:", a ** b)  # 10^3
```

```
Addition: 13
Subtraction: 7
Multiplication: 30
Division: 3.3333333333333335
Floor Division: 3
Modulus: 1
Exponentiation: 1000
```


Python uses the standard PEMDAS **order of operations** to perform calculations

1. **P**arentheses
2. **E**xponentiation
3. **M**ultiplication & **D**ivision *(including Floor Division & Modulo), from left to right*
4. **A**ddition & **S**ubtraction, from left to right

**Without Parentheses**


**With Parentheses** *(to control execution)*




**Real-life Use Case:**

Calculating totals, discounts, or percentages in e-commerce.

**Comparison Operators:**

Used to compare values: ==, !=, >, <, >=, <=.

```
# Example: Comparison Operators
x = 10
y = 5
print("Is x equal to y?", x == y)
print("Is x greater than y?", x > y)
print("Is x less than or equal to y?", x <= y)
```

```
Is x equal to y? False
Is x greater than y? False
Is x less than or equal to y? True
```

**Real-life Use Case:**

Validating user input or checking conditions in a program.

**Logical Operators:**

Used to combine conditions: and, or, not.

```
# Example: Logical Operators
is_student = True
is_working = False
print("Is student and working?", is_student and is_working)
print("Is student or working?", is_student or is_working)
print("Is not working?", not is_working)
```

```
Is student and working? False
Is student or working? True
Is not working? True
```

**Real-life Use Case:**

Checking multiple conditions (e.g., "Is the user a student and under 18?").

## 4. Numeric & String Functions

**Numeric Functions:**

**abs()**: Absolute value.

**round()**: Round a number.

**min(), max()**: Find minimum or maximum in a list.

```
# Example: Numeric Functions
print("Absolute value of -10:", abs(-10))
print("Rounded value of 3.14159:", round(3.14159, 2))  # Round to 2 decimal places
print("Minimum of [5, 2, 8]:", min([5, 2, 8]))
print("Maximum of [5, 2, 8]:", max([5, 2, 8]))
```

```
Absolute value of -10: 10
Rounded value of 3.14159: 3.14
Minimum of [5, 2, 8]: 2
Maximum of [5, 2, 8]: 8
```

**Real-life Use Case:**

Calculating absolute differences or rounding prices.

**String Functions:**

**len()**: Length of a string.

**str()**: Convert to string.

```
# Example: String Functions
text = "Hello, World!"
print("Length of text:", len(text))
print("Number as string:", str(100))
```

```
Length of text: 13
Number as string: 100
```

**Real-life Use Case:**

Validating the length of a password or converting numbers to text for display.

## 5. String Indexing & Slicing:

**Indexing:** Access individual characters using [index].

**Slicing:** Extract a substring using [start:end:step].

```
# Example: String Indexing & Slicing

text = "Python Programming"
print("First character:", text[0])  # Indexing
print("Substring from index 7 to 11:", text[7:12])  # Slicing
print("Every second character:", text[::2])  # Step
print("Reverse string:", text[::-1])  # Reverse

First character: P
Substring from index 7 to 11: Progr
Every second character: Pto rgamn
Reverse string: gnimmargorP nohtyP
```

**Real-life Use Case:**

Extracting parts of a URL or processing text data.

## 6. String Methods:

**Common Methods:**

**.lower()**: Convert to lowercase.

**.upper()**: Convert to uppercase.

**.strip()**: Remove leading/trailing whitespace.

**.replace()**: Replace substrings.

**.split()**: Split into a list of substrings.

**.join()**: Join a list into a string.

```
# Example: String Methods

text = "  Hello, World!  "
print("Lowercase:", text.lower())
print("Uppercase:", text.upper())
print("Stripped:", text.strip())
print("Replaced:", text.replace("World", "Python"))
print("Split:", text.split(","))  # Split at comma
print("Joined:", "-".join("2025","10","15"))  # Join with hyphen

Lowercase:  hello, world!
Uppercase:  HELLO, WORLD!
Stripped: Hello, World!
Replaced:  Hello, Python!
Split: ['  Hello', ' World!  ']
Joined: 2025-10-15
```

**Real-life Use Case:**

Cleaning user input or formatting text for display.

## 7. F-Strings:

**F-Strings:** Formatted string literals for embedding expressions inside strings.

**Syntax:** f"Text {expression}".

```
# Example: F-strings

name = "Alice"
age = 25
print(f"My name is {name} and I am {age} years old.")
print(f"Next year, I will be {age + 1} years old.")
```

```
My name is Alice and I am 25 years old.
Next year, I will be 26 years old.
```

**Real-life Use Case:**

Generating dynamic messages or reports.

## 8. Practice Exercise:

- Create a program that takes two numbers as input and performs all arithmetic operations.
- Write a program to reverse a string and check if it is a palindrome.
- Use f-strings to create a dynamic sentence using user input (name, age, and favorite color).