# Python Fundamentals: A Comprehensive Tutorial

This notebook covers 7 essential Python concepts:

1. Variables
2. Type Casting
3. User Input/Output
4. Arithmetic and Math Functions
5. Conditional Logic
6. Strings
7. Loops & Collections

## 1. Variables

**Definition:** Variables are named containers that contain data values in your program. You can place any type of data in a variable (numbers, text, true/false). Once created, you can access or change the data by it's name.

```
# Variables are containers for storing data values
name = "Alice"
age = 25
gpa = 3.85
is_student = True

print(f"Name: {name}, Age: {age}, GPA: {gpa}, Student: {is_student}")

# Multiple assignment
x, y, z = 10, 20, 30
print(f"x={x}, y={y}, z={z}")
```

## 2. Type Casting

**Definition:** Type casting converts a value from one data type to another. Python provides functions like int(), str(), and float() to convert between types. This is useful when you need to change how data is stored or displayed.

```
# String to Integer
num_str = "42"
num_int = int(num_str)
print(f"String '{num_str}' converted to int: {num_int}")

# Integer to String
number = 100
```

```
text = str(number)
print(f"Integer {number} converted to string: '{text}'")

# String to Float
price_str = "19.99"
price_float = float(price_str)
print(f"String '{price_str}' converted to float: {price_float}")

# Float to Integer
value = float(7.9)
converted = int(value)
print(f"Float {value} converted to int: {converted}")
```

## 3. User Input/Output

**Definition:** Input allows your program to receive data from the user via keyboard using the input() function. Output displays information back to the user on the screen using print(). Together, they enable interaction between the program and the user.

```
# Basic input/output
print("=== User Input/Output Example ===")
user_name = input("Enter your name: ")
user_age = input("Enter your age: ")
user_age = int(user_age)  # Convert to integer

print(f"\nHello, {user_name}!")
print(f"You are {user_age} years old.")
```

## 4. Arithmetic and Math Functions

**Definition:** Arithmetic operations perform mathematical calculations like addition, subtraction, multiplication, and division using operators (+, -, *, /). Python also has a math module with built-in functions like sqrt(), abs(), and pow(). These operations and functions make Python useful for any calculation-based tasks.

```
import math

# Basic arithmetic operations
a, b = 15, 4

addition = a + b
subtraction = a - b
multiplication = a * b
division = a / b
floor_division = a // b
modulus = a % b
exponent = a ** b
```

```
print("\nMath Functions:")
print(f"Square root of 16: {math.sqrt(16)}")
print(f"Absolute value of -25: {abs(-25)}")
print(f"Maximum of 5, 12, 3: {max(5, 12, 3)}")
print(f"Minimum of 5, 12, 3: {min(5, 12, 3)}")
print(f"Power 2^8: {pow(2, 8)}")
print(f"Round 3.7: {round(3.7)}")
```

## 5. Conditional Logic

**Definition:** Conditional logic lets your program make decisions based on certain conditions. You use if/elif/else statements to run different code depending on whether a condition is true or false. Logical operators (and, or, not) help combine multiple conditions to create complex decision-making.

```
# If-elif-else statements
score = 85

if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
else:
    grade = "F"

print(f"Score: {score}, Grade: {grade}")

# Logical operators
age = 20
has_license = True

if age >= 18 and has_license:
    print("You can drive!")
elif age >= 18 or has_license:
    print("You meet some requirements")
else:
    print("You cannot drive")

# NOT operator
is_raining = False
if not is_raining:
    print("It's a nice day for a walk!")

# Ternary/Conditional expression
status = "Pass" if score >= 70 else "Fail"
print(f"Status: {status}")
```

# 6. Strings

**Definition:** Strings are sequences of characters (letters, numbers, symbols) enclosed in quotes. Strings have methods (like upper(), lower(), replace()) that let you manipulate text in different ways. You can access individual characters by position (indexing) or extract portions using slicing.

```
text = "Python Programming"

# String methods
print("String Methods:")
print(f"Original: {text}")
print(f"Uppercase: {text.upper()}")
print(f"Lowercase: {text.lower()}")
print(f"Length: {len(text)}")
print(f"Replace: {text.replace('Python', 'Java')}")
print(f"Contains 'Pro': {'Pro' in text}")

# String indexing (0-based)
print("\nString Indexing:")
print(f"First character: {text[0]}")
print(f"Last character: {text[-1]}")
print(f"Slice [0:6]: {text[0:6]}")
print(f"Every 2nd character: {text[::2]}")

# String formatting
name = "Bob"
age = 30
score = 95.5

print("\nString Formatting:")
# f-string (modern approach)
print(f"F-string: {name} is {age} years old with score {score:.1f}")

# .format() method
print("Format method: {} is {} years old".format(name, age))
```

# 7. Loops & Collections

**Definition:** Loops let you loop and repeat code multiple times without writing it over and over. Collections are containers that hold multiple values: lists, tuples, sets, dictionaries, and 2D lists.

```
print("--- Loops ---\n")

# While loop
print("While loop (count 1-3):")
count = 1
while count <= 3:
    print(f"  Count: {count}")
```

```
    count += 1

# For loop with range
print("\nFor loop (0-4):")
for i in range(5):
    print(f"  i = {i}")

# For loop with list
print("\nFor loop with list:")
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(f"  {fruit}")

# Nested loops
print("\nNested loops (multiplication table):")
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i}*{j}={i*j}", end="  ")
    print()

# Lists - ordered, mutable
print("Lists (ordered, can be changed):")
numbers = [10, 20, 30, 40, 50]
numbers.append(60)
numbers.insert(0, 5)
print(f"  List: {numbers}")
print(f"  First element: {numbers[0]}")
print(f"  Slice [1:3]: {numbers[1:3]}")

# Tuples - ordered, immutable
print("\nTuples (ordered, cannot be changed):")
coordinates = (10, 20, 30)
print(f"  Tuple: {coordinates}")
print(f"  First: {coordinates[0]}")

# Sets - unordered, unique values
print("\nSets (unordered, unique values only):")
unique_nums = {1, 2, 3, 2, 1}
print(f"  Set: {unique_nums}")

# Dictionaries - key-value pairs
print("\nDictionaries (key-value pairs):")
student = {
    "name": "John",
    "age": 20,
    "gpa": 3.8,
    "courses": ["Math", "Physics"]
}
print(f"  Name: {student['name']}")
print(f"  Age: {student['age']}")
```

```python
print(f"  All data: {student}")

# 2D Lists
print("\n2D Lists (grids or matrices):")
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
print(f"  Matrix: {matrix}")
print(f"  Element at row 1, column 2: {matrix[1][2]}")
```

# Reflection

## Most Difficult Part

The hardest thing to grasp was **type casting and interaction between data types**. At first, I found it hard to know when and why conversions were needed. For instance, user input from `input()` is always a string, so the need to convert it to an integer with `int()` wasn't clear at first. I got past this by running several tiny programs that intentionally messed around with types and saw the errors. That practical experimentation made the concept concrete, not abstract. I also came to understand that dynamic typing is both an asset (flexible) and liability (needs to be mindful about types) for Python.

## Future Application Plans

In future academic projects, I will utilize Python greatly for **data analysis** with pandas and numpy libraries. For a stats course, I can automate the processing and visualization of the data. In personal projects, I find interest in creating **automation scripts**—managing files, web scraping, or simple games such as the quiz example above. The reason for using Python is due to the fact that it is very easy for rapid prototyping. I also see promise for **machine learning** projects with scikit-learn for predictive models. The basic concepts learned here—particularly loops, collections, and conditional statements—provide the foundation for all these higher-end applications. I believe that the mastery of these basics will greatly propel me toward more specialized domains in Python.

## Citations

Bro Code. "Start coding with PYTHON in 5 minutes! ." YouTube, 27 Oct. 2025, www.youtube.com/watch?v=XKHEtdqhLK8 Gemini2.5Pro "I am injured, please help create caption for code..."prompt 27 Oct. 2025, Google.Inc