Basic Python

Prof. Pranay Kumar Saha

October 8, 2025

Why Python?

Advantages:

- Easy to learn
- Readable syntax
- Versatile applications
- Large community
- Rich libraries

Popular Uses:

- Web development
- Data science
- Automation
- AI/ML
- Scripting

Hello World

```
# This is a comment
print("Hello, World!")

# Multi-line comment
"""
This is a
multi-line comment
"""
```

- Python uses **#** for single-line comments
- Triple quotes for multi-line strings/comments
- No semicolons required

Variables and Data Types

```
# Dynamic typing - no declaration needed
name = "Alice"            # string
age = 25                  # integer
height = 5.6              # float
is_student = True         # boolean

# Multiple assignment
x, y, z = 1, 2, 3

# Type conversion
num_str = "42"
num_int = int(num_str)
```

Basic Data Types

```
# Numbers
integer = 42
floating = 3.14
complex_num = 3 + 4j

# Strings
single = 'Hello'
double = "World"
multi = """Multiple
lines"""

# Boolean
is_true = True
is_false = False
```

Raw Strings (r-strings)

Prefixed with **r**. In raw strings, backslashes (****) are treated as literal characters, not as escape sequences. This is very useful for file paths and regular expressions.

```
# Standard string interprets \n as a newline
path1 = 'C:\Users\new_folder'
print(f"Standard: {path1}")

# Raw string treats \n as literal characters
path2 = r'C:\Users\new_folder'
print(f"Raw: {path2}")
```

Formatted Strings (f-strings)

Prefixed with **f**. They allow you to embed Python expressions directly inside string literals for easy formatting.

```
name = "Alice"
age = 30

# Embed variables directly into the string
greeting = f"Hello, my name is {name}."
print(greeting)

# You can also evaluate expressions inside
info = f"{name} will be {age + 5} in five years."
print(info)
```

Byte Strings (b-strings)

Prefixed with 'b'. A byte string is a sequence of bytes (integers from 0-255), not Unicode characters. They are used for binary data, network communication, or file I/O.

```
# A regular string (Unicode)
unicode_string = "hello"
print(type(unicode_string))

# A byte string (sequence of bytes)
byte_string = b"hello"
print(type(byte_string))
print(byte_string)

# Each character corresponds to an ASCII value
print(byte_string[0]) # Prints the ASCII value of 'h'
```

Indentation Matters!

```
# Python uses indentation for code blocks
if age >= 18:
    print("Adult")
    print("Can vote")
else:
    print("Minor")
    print("Cannot vote")

# Indentation error
if True:
print("This will cause an error!")
```

Use 4 spaces (PEP 8 standard) for indentation

Operators

Arithmetic:

```
+ - * /
// (floor division)
% (modulus)
** (exponent)
```

Comparison:

```
== != < > <= >=
```

Logical:

```
and or not
```

Assignment:

```
= += -= *= /=
//= %= **=
```

Control Flow - If Statements

```
score = 85

if score >= 90:
    print("Grade: A")
elif score >= 80:
    print("Grade: B")
elif score >= 70:
    print("Grade: C")
else:
    print("Grade: F")

# Ternary operator
result = "Pass" if score >= 60 else "Fail"
```

Loops - For Loop

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
   print(fruit)

for i in range(5): # 0 to 4
   print(i)

for i in range(2, 10, 2): # 2,4,6,8
   print(i)

for idx, fruit in enumerate(fruits):
   print(f"{idx}: {fruit}")
```

Loops - While Loop

```
# While loop
count = 0
while count < 5:
   print(count)
   count += 1

# Break and continue
for i in range(10):
   if i == 3:
      continue # Skip 3
   if i == 7:
      break # Stop at 7
   print(i)
```

Data Structures - Lists

```
# Lists are mutable, ordered collections
numbers = [1, 2, 3, 4, 5]
mixed = [1, "two", 3.0, True]

# Operations
numbers.append(6) # Add to end
numbers.insert(0, 0) # Insert at index
numbers.remove(3) # Remove value
popped = numbers.pop() # Remove and return last

# Slicing
first_three = numbers[0:3] # [0, 1, 2]
last_two = numbers[-2:] # Last 2 items
```

Data Structures - Tuples and Sets

```
# Tuples - immutable, ordered
coordinates = (10, 20)
single = (1,) # Note the comma

# Sets - unordered, unique elements
unique_nums = {1, 2, 3, 3, 4} # {1,2,3,4}
set_ops = {1, 2} | {2, 3} # Union: {1,2,3}
set_ops = {1, 2} & {2, 3} # Intersection: {2}
set_ops = {1, 2} - {2, 3} # Difference: {1}
```

Data Structures - Dictionaries

```
person = {
   "name": "Alice",
   "age": 25,
   "city": "New York"
}

print(person["name"])
person["age"] = 26
person["email"] = "alice@email.com"

keys = person.keys()
values = person.values()
items = person.items()
```

String Operations

```
text = "Python Programming"

lower = text.lower()
upper = text.upper()
title = text.title()
split = text.split() # Split by space
joined = "-".join(["a","b"]) # "a-b"

name = "Alice"
age = 25
message = f"{name} is {age} years old"

message = "{} is {} years old".format(name, age)
```

File Handling

```
# Reading files
with open("file.txt", "r") as f:
   content = f.read()
   # File automatically closed

# Reading line by line
with open("file.txt", "r") as f:
   for line in f:
      print(line.strip())

# Writing files
with open("output.txt", "w") as f:
   f.write("Hello, World!\n")
   f.writelines(["Line 1\n", "Line 2\n"])
```

Basic Python

Thank You for Listening!