Introduction to Python Basics

Lecture Slides

March 13, 2025

What is Python?

Python Overview

Description: Python is a high-level, interpreted language known for its simplicity and readability.

- Created by Guido van Rossum, first released in 1991.
- Supports multiple paradigms: object-oriented, procedural, functional.
- Features a vast ecosystem of libraries for data analysis, web development, and more.

Comparing Python with C and C++

Conceptual Comparison

- *Python*:
  - Simple, concise syntax.
  - Interpreted: Runs without compilation.
  - Dynamic typing: No need to declare variable types.
- *C*:
  - Low-level, closer to hardware.
  - Compiled: Requires compilation to machine code.
  - Manual memory management (e.g., malloc, free).
- *C++*:
  - Compiled, supports object-oriented programming.
  - Complex syntax with templates and classes.
  - Manual and automatic memory management (e.g., new, delete).

Code Comparison: ”Hello, World”

Example: Printing ”Hello, World”

```
# Python
print("Hello,␣World")

// C
#include <stdio.h>
int main() {
    printf("Hello,␣World\n");
    return 0;
}

// C++
#include <iostream>
int main() {
    std::cout << "Hello,␣World" << std::endl;
    return 0;
}
```

Values, Expressions, and Statements

Values: Fundamental data pieces in Python.

```
42          # Integer value
3.14        # Float value
"Hello"     # String value
True        # Boolean value
```

Expressions: Combination of values and operators.

```
5 + 3 * 2
"Hello" + "␣World"
4 > 2
```

Statements: Instructions Python executes.

```
x = 10
y = x * 2
print(y)
```

Numbers, Booleans, and Strings

Numbers (Integers and Floats):

```
x = 10       # integer
y = 10.5     # float
```

Booleans (True or False):

```
is_valid = True
has_error = False
```

Strings (Text data):

```
name = "Alice"
message = ’Hello,␣World!’
```

Operators

Arithmetic Operators:

```
x = 10 + 5
x = 10 - 5
x = 10 * 2
x = 10 / 2
x = 10 % 3  # modulo operator
```

Comparison Operators:

```
10 > 5   # True
10 < 5   # False
10 == 10 # True
10 != 5  # True
```

Logical Operators:

```
(10 > 5) and (3 > 1)  # True
(10 < 5) or (3 > 1)   # True
not (10 < 5)          # True
```

Variables and Keywords

Variables: Containers for storing data.

```python
age = 25
price = 19.99
name = "Bob"
print(age, price, name)
```

Keywords: Reserved words in Python.

```python
# Examples of Python keywords:
if, else, for, while, def, return,
import, from, as, True, False, None
```

Note: You can't use keywords as variable names.

Example

Example of Python code:

```
# Calculate area of a circle
radius = 5
pi = 3.14159
area = pi * radius ** 2

print(f"The␣area␣is␣{area:.2f}")
```

Output:

```
The area is 78.54
```

A String Basics

Strings are sequences of characters in Python.

```
text = "Hello,␣World!"
print(text[0])       # 'H'
print(len(text))     # 13
```

Concatenation and Repetition

Concatenation (+):

```
greet = "Hello" + "␣" + "World"
print(greet) # Hello World
```

Repetition (*):

```
repeat = "Hi!␣" * 3
print(repeat) # Hello Hello Hello
```

Slicing and Indexing

Slicing syntax: `string[start:end:step]`

```
text = "Hello␣World"
print(text[0:5])     # Hello
print(text[6:])      # World
print(text[:5])      # Hello
print(text[::-1])    # dlroW olleH (Reverse)
```

String Methods: Searching

```
text = "Python␣is␣fun"

text.find("is")      # returns index 7
text.index("fun")    # raises ValueError if not found
text.count("n")      # counts occurrences
```

String Methods for Modification

Case manipulation:

```
"hello".upper()        # 'HELLO'
"HELLO".lower()        # hello
"python␣is␣fun".title()  # Python Is Fun
```

Strip whitespace:

```
text = "␣␣hello␣␣"
text.strip()     # 'hello'
text.lstrip()    # 'hello  '
text.rstrip()    # '  hello'
```

Checking String Content

```python
text = "Hello123"

text.isalpha()    # False (contains spaces)
text.isdigit()    # False
text.isalnum()   # False (because of space)

"Hello".isalpha() # True
"123".isdigit()   # True
```

Replace and Split

Replace substrings:

```python
text = "I␣like␣cats"
text.replace("cats", "dogs")  # 'I like dogs'
```

Split and Join:

```python
text = "apple,banana,cherry"
fruits = text.split(",")
# ['apple', 'banana', 'cherry']

new_text = "-".join(fruits)
# 'apple-banana-cherry'
```

Formatting Strings

Using format method:

```python
name = "Alice"
age = 30
message = "{}␣is␣{}␣years␣old.".format(name, age)
print(message)
# Alice is 25
```

Using f-strings (Python 3.6+):

```python
name = "Alice"
age = 25
print(f"{name}␣is␣{age}")
# Alice is 25
```

Checking and Case Conversion

Checking substrings:

```
text = "Hello␣World"
print("World" in text) # True
```

Changing case:

```
text = "hello"
text.upper()  # 'HELLO'
text.lower()  # 'hello'
text.capitalize() # 'Hello'
text.title()      # 'Hello'
```

Formatting and Padding

String alignment:

```
text = "Python"
text.ljust(10, "-")  # ’Python----’
text.rjust(10, "-")  # ’----Python’
text.center(10, "-") # ’--Python--’
```

Padding zeros:

```
"42".zfill(5)  # ’00042’
```

User Input

Using `input()` to get user input:

```
name = input("Enter␣your␣name:␣")
print(f"Hello,␣{name}!")
```

Note:

- `input()` always returns a string.

Type Casting (Conversion)

Convert data from one type to another explicitly.

```
age = input("Enter␣your␣age:␣")
age = int(age)    # Convert string to integer
print(age + 10)
```

Common Casting Functions:

```
int("123")     # 123
float("12.5")  # 12.5
str(123)       # "123"
bool(0)        # False
```

Comments in Python

Comments are used to explain code.

Single-line comments:

```python
# This is a single-line comment
x = 10  # variable initialization
```

Multi-line comments (Docstrings):

```python
'''
This is a
multi-line comment
used as a docstring.
'''

def greet():
    """This function greets the user."""
    print("Hello!")
```

Practical Example

Combine input, casting, and comments:

```python
# Get user's birth year
birth_year = input("Enter␣your␣birth␣year:␣")

# Calculate user's age
current_year = 2025
age = current_year - int(birth_year)

# Display the result
print(f"You␣are␣{age}␣years␣old.")
```

Sample Output:

```
Enter your birth year: 2000
You are 25 years old.
```

Lists in Python

Lists are mutable (modifiable), ordered collections of items.

Creating lists:

```python
numbers = [1, 2, 3, 4]
fruits = ["apple", "banana", "cherry"]
mixed = [1, "apple", 3.14, True]
```

Accessing items:

```python
print(fruits[0])  # First item
print(fruits[-1]) # Last item
```

List Operations

Adding items:

```
fruits = ["apple", "banana"]
fruits.append("cherry")
# [’apple’, ’banana’, ’cherry’]
```

Removing items:

```
fruits.remove("banana") # Remove by value
fruits.pop(1)            # Remove by index
```

Slicing lists:

```
numbers = [1, 2, 3, 4, 5]
print(numbers[1:4]) # [2, 3, 4]
```

List Methods

Common methods:

```python
numbers = [1, 2, 3]
numbers.append(4)    # [1,2,3,4]
numbers.extend([5,6]) # [1,2,3,4,5,6]
numbers.insert(0, 0) # [0,1,2,3,4,5,6]
numbers.sort()       # sorts list
numbers.reverse()    # reverses order
```

Length of lists:

```python
len(numbers)  # 6
```

Tuples in Python

Tuples are immutable (unchangeable), ordered collections of items.

Creating tuples:

```python
coordinates = (10.0, 20.5)
colors = ("red", "green", "blue")
```

Accessing items:

```python
print(colors[0])  # First item
print(colors[-1]) # Last item
```

Tuple Operations

Tuple unpacking:

```python
coordinates = (10.0, 20.0)
x, y = coordinates
print(x)  # 10
print(y)  # 20
```

Concatenation and repetition:

```python
tuples = (1, 2) + (3, 4)  # (1, 2, 3, 4)
tuples_repeated = (1,) * 3  # (1, 1, 1)
```

Note: Tuples don't support methods like append, remove, etc.

Lists vs Tuples

Lists (mutable):

```
mutable_list = [1, 2, 3]
mutable[0] = 5  # Works fine
```

Tuples (Immutable):

```
immutable = (1, 2, 3)
immutable[0] = 5  # Raises TypeError
```

When to use?

- Lists for modifiable data.
- Tuples for fixed data.

Dictionaries in Python

Dictionaries store key-value pairs (mutable).

Creating dictionaries:

```python
student = {"name": "Alice", "age": 24}
```

Accessing values:

```python
print(student["name"])  # Alice
```

Adding/updating entries:

```python
student["grade"] = "A"
student["age"] = 25
```

Dictionary Operations

Common methods:

```python
student = {"name": "Alice", "age": 24}

student.keys()      # dict_keys([’name’, ’age’])
student.values()    # dict_values([’Alice’, 24])
student.items()     # dict_items([(’name’,’Alice’), (’age’,24)])

student.get("name") # ’Alice’
student.get("grade", "N/A") # ’N/A’
```

Adding and removing:

```python
student["grade"] = ’A’
del student["age"]
```

Sets in Python

Sets store unique, unordered collections of items.

Creating sets:

```python
fruits = {"apple", "banana", "cherry"}
```

Sets automatically remove duplicates:

```python
numbers = {1, 2, 2, 3, 3, 3}
print(numbers)  # {1, 2, 3}
```

Set Operations

Adding and removing items:

```
numbers = {1, 2, 3}
numbers.add(4)      # {1,2,3,4}
numbers.remove(2)   # {1,3,4}
```

Set operations:

```
a = {1, 2, 3}
b = {3, 4, 5}

a | b  # Union: {1,2,3,4,5}
a & b  # Intersection: {3}
a - b  # Difference: {1,2}
a ^ b # Symmetric difference: {1,2,4}
```

Dictionary vs. Set

Dictionary (key-value pairs):

```python
student = {"name": "Alice", "age": 24}
print(student["name"]) # Alice
```

Set (unique items, no keys):

```python
numbers = {1, 2, 3, 4}
numbers.add(5)
```

Usage:

- Dictionaries: store related data with clear labels.
- Sets: unique items, mathematical operations.

Control Flow: if-elif-else Statements

if-elif-else Statements

- Conditional execution based on multiple conditions.
- Use if for the first condition, elif for additional conditions, and else for the default case.

```python
age = 20
if age < 18:
    print("Minor")
elif age == 18:
    print("Exactly 18")
else:
    print("Adult")
# Output: Adult
```

Control Flow: Nested Conditionals

Nested Conditionals

- Conditions within conditions for complex logic.
- Inner if statements execute only if outer conditions are true.

```python
age = 20
if age >= 18:
    if age >= 21:
        print("Adult,␣21␣or␣older")
    else:
        print("Adult,␣under␣21")
else:
    print("Minor")
# Output: Adult, under 21
```

Looping Structures

For Loop:

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

Looping Structures. (Cont.)

While Loop:

```
count = 0
while count < 3:
    print(count)
    count += 1
# Output:
# 0
# 1
# 2
```

Loop Control Statements: Break

break: Exit loop immediately.

```
for num in range(5):
    if num == 3:
        break
    print(num)
# Output: 0 1 2
```

Loop Control Statements: Continue

continue: Skip current iteration.

```python
for num in range(5):
    if num == 2:
        continue
    print(num)
# Output: 0 1 3 4
```

Loop Control Statements: Pass

pass: Placeholder, does nothing.

```python
if True:
    pass  # Placeholder for future code
```

List Comprehensions

Concise syntax for creating lists:

Basic syntax:

```python
squares = [x**2 for x in range(5)]
print(squares)  # [0, 1, 4, 9, 16]
```

Conditional comprehension:

```python
evens = [x for x in range(10) if x % 2 == 0]
print(evens)  # [0, 2, 4, 6, 8]
```

Nested List Comprehensions

Example of nested loops simplified:

```
pairs = [(x, y) for x in range(3) for y in ’ab’]
print(pairs)
# Output: [(0,’a’), (0,’b’), (1,’a’), (1,’b’), (2,’a’), (2,’b’)]
```

Matrix creation using nested comprehension:

```
matrix = [[i * j for j in range(3)] for i in range(3)]
print(matrix)
# Output: [[0,0,0],[0,1,2],[0,2,4]]
```

Set and Dictionary Comprehensions

Set comprehension:

```python
square_set = {x**2 for x in range(5)}
print(square_set)  # {0, 1, 4, 9, 16}
```

Dictionary comprehension:

```python
square_dict = {x: x**2 for x in range(5)}
print(square_dict)
# Output: {0:0, 1:1, 2:4, 3:9, 4:16}
```

Conditional dictionary comprehension:

```python
even_squares = {x: x**2 for x in range(5) if x % 2 == 0}
print(even_squares)
# Output: {0:0, 2:4, 4:16}
```

Practical Example: Comprehensions

```python
scores = [65, 72, 85, 40, 90]
passed = [score for score in scores if score >= 60]

grades = {score: (’Pass’ if score >= 60 else ’Fail’) for score in scores}

print(passed) # [65,72,85,90]
print(grades)
# Output: {65:’Pass’, 72:’Pass’, 85:’Pass’, 40:’Fail’, 90:’Pass’}
```

Set and Dictionary Comprehensions

Set comprehension example:

```python
squares_set = {x**2 for x in range(5)}
print(squares_set)
# Output: {0, 1, 4, 9, 16}
```

Dictionary comprehension example:

```python
square_dict = {x: x**2 for x in range(5)}
print(square_dict)
# Output: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Iterators and Iterables

Iterator basics:

```python
fruits = ["apple", "banana"]
fruit_iter = iter(fruits)

print(next(fruit_iter))  # apple
print(next(fruit_iter))  # banana
```

Looping through dictionaries:

```python
student = {"name": "Alice", "age": 24}
for key, value in student.items():
    print(f"{key}:␣{value}")
# Output:
# name: Alice
# age: 24
```

Practical Control Flow Example

Real-world scenario:

```python
scores = [85, 42, 78, 90, 67]
passed = [score for score in scores if score >= 60]

for score in passed:
    if score > 80:
        print(f"Excellent:␣{score}")
    else:
        print(f"Good:␣{score}")

# Output:
# Excellent: 85
# Good: 78
# Excellent: 90
# Good: 67
```