ENSF 381
Full Stack Web Development

Lecture 26: Python

Slides: Ahmad Abdellatif, PhD

Instructor: Novarun Deb, PhD

UNIVERSITY OF CALGARY

Outline

- Introduction to Python.
- Variables.
- Strings.
- Conditional statements.
- Loops.

Python

- Created by Guido van Rossum in the late 1980s.
- The language was designed with the idea that code should be easy to write and understand.
- An open-source language, meaning its source code is freely available to the public.
- Python has gone through several major versions, with the most notable being Python 2 and Python 3.

Key features

- **Easy to Learn and Read**: Python's syntax is designed to be clear and readable, making it easy for beginners to learn and understand.

- **Rapid Prototyping**: Python is often used for rapid prototyping and development due to its simplicity and readability, enabling faster iterations in the development process.

- **Large Standard Library**: Python comes with a comprehensive standard library that includes modules and packages for a wide range of tasks, reducing the need for external libraries.

- **Cross-Platform Compatibility**: Python is a cross-platform language, meaning that code written on one platform (e.g., Windows) can run on another (e.g., Linux or macOS) without modification.

- **Data Analysis and Exploration**: widely used for exploratory data analysis. Also, Python provides robust tools for building and training machine learning models.

Variables

- Variables are used to store and reference data values in the program.
- Python is dynamically typed, eliminating the need to declare the data type of variables explicitly.
- Variable names are **case-sensitive**.
- Comments starts with #, and Python will ignore them.

First Python example

```python
# Variables and data types
age = 25
height = 1.75
name = "John"
is_student = True
weight = 60
```

First Python example

```
26
19.591836734693878
(.venv) ahmadabdellatif@itadmins-MacBook-Pro workspace %
```

Strings

- Strings can be enclosed by either single quotes or double quotes.

```
course = 'Welcome to ENSF381!'
course = "Welcome to ENSF381!"
```

- In Python, strings are considered as arrays.

- Strings can be concatenated **exclusively** using the + operator with another string.

String concatenation

```python
course = 'Welcome to ENSF' + 381
print(course)
```

```
Traceback (most recent call last):
  File "/Users/ahmadabdellatif/workspace/test/sample.py", line 6, in <module>
    course = 'Welcome to ENSF' + 381
             ~~~~~~~~~~~~~~~~~~~~^~~~
TypeError: can only concatenate str (not "int") to str
```

```python
course = 'Welcome to ENSF' + "381"
print(course)
```

```
Welcome to ENSF381
```

```python
course = 'Welcome to ENSF' + str(381)
print(course)
```

```
Welcome to ENSF381
```

A built-in function that converts a specified value into a string.

String useful methods

- `format`: a method used for string formatting, allowing the insertion of values into a string template.

String useful methods

Code Snippet

```
name = "John"
age = 25
message = "My name is {} and I am {} years old.".format(name, age)
print(message)
```

Output

My name is John and I am 25 years old.

String useful methods

- `format`: a method used for string formatting, allowing the insertion of values into a string template.

- `split`: a method applied to strings that divides the string into a list of substrings based on a specified delimiter.

String useful methods

Code Snippet

```
name = "John"
age = 25
message = "My name is {} and I am {} years old.".format(name, age)
print(message)
```

Output

My name is John and I am 25 years old.

```
sentence = "Hello, world!"
words = sentence.split(", ")
print(words)
```

['Hello', 'world!']

String useful methods

- `format`: a method used for string formatting, allowing the insertion of values into a string template.

- `split`: a method applied to strings that divides the string into a list of substrings based on a specified delimiter.

- `strip`: a method used to remove leading and trailing whitespaces (including newline characters) from a string.

String useful methods

Code Snippet

```
name = "John"
age = 25
message = "My name is {} and I am {} years old.".format(name, age)
print(message)
```

Output

My name is John and I am 25 years old.

```
sentence = "Hello, world!"
words = sentence.split(", ")
print(words)
```

['Hello', 'world!']

```
text = "   This is a string with whitespace.   "
clean_text = text.strip()
print(clean_text)
```

This is a string with whitespace.

String useful methods

- `format`: a method used for string formatting, allowing the insertion of values into a string template.

- `split`: a method applied to strings that divides the string into a list of substrings based on a specified delimiter.

- `strip`: a method used to remove leading and trailing whitespaces (including newline characters) from a string.

- `find`: a method that searches for a specified substring within a string and returns the index of the first occurrence. If not found, it returns -1.

String useful methods

Code Snippet | **Output**

```
name = "John"
age = 25
message = "My name is {} and I am {} years old.".format(name, age)
print(message)
```

→ My name is John and I am 25 years old.

```
sentence = "Hello, world!"
words = sentence.split(", ")
print(words)
```

→ ['Hello', 'world!']

```
text = "   This is a string with whitespace.   "
clean_text = text.strip()
print(clean_text)
```

→ This is a string with whitespace.

```
sentence = "Python is powerful and Python is easy to learn."
index = sentence.find("Python")
print(index)
```

→ 0

String useful methods

- `format`: a method used for string formatting, allowing the insertion of values into a string template.

- `split`: a method applied to strings that divides the string into a list of substrings based on a specified delimiter.

- `strip`: a method used to remove leading and trailing whitespaces (including newline characters) from a string.

- `find`: a method that searches for a specified substring within a string and returns the index of the first occurrence. If not found, it returns -1.

- `replace`: a method that replaces occurrences of a specified substring with another substring in a string.

String useful methods

Code Snippet

```
name = "John"
age = 25
message = "My name is {} and I am {} years old.".format(name, age)
print(message)
```

→ My name is John and I am 25 years old.

```
sentence = "Hello, world!"
words = sentence.split(", ")
print(words)
```

→ ['Hello', 'world!']

```
text = "   This is a string with whitespace.   "
clean_text = text.strip()
print(clean_text)
```

→ This is a string with whitespace.

```
sentence = "Python is powerful and Python is easy to learn."
index = sentence.find("Python")
print(index)
```

→ 0

```
sentence = "Python is fun!"
new_sentence = sentence.replace("fun", "awesome")
print(new_sentence)
```

→ Python is awesome!

Output

Conditional statements

- Conditional statements in Python (if, elif, else) are similar to those in many other programming languages.

- Python uses **indentation** to define code blocks (no braces).

- Comparison operators:
  - Equals (==).
  - Not Equals (!=).
  - Greater Than (>), Greater Than or Equal To (>=).
  - Less Than (<), Less Than or Equal To (<=).

- Logical operators:
  - and: returns True if both conditions are True.
  - or: returns True if at least one condition is True.
  - not: returns True if the condition is False and vice versa.

Conditional statements - syntax

```
if condition:
    # Code to be executed if the condition is True
elif another_condition:
    # Code to be executed if another_condition is True
elif yet_another_condition:
    # Code to be executed if yet_another_condition is True
else:
    # Code to be executed if none of the conditions are True
```

- If the initial condition is not met, the program will check another_condition.
- If another_condition is also not met, it will then check yet_another_condition.
- The else block will be executed only if none of the conditions are True.

Conditional statements - example

```
temperature = 28
is_sunny = True

# Conditional statements based on variables
if temperature > 25 and is_sunny:
  print("It's a hot and sunny day!")

elif temperature > 25 and not is_sunny:
  print("It's warm but not sunny.")

elif temperature <= 25 and is_sunny:
  print("It's a cool day with sunshine.")

else:
  print("It's a cool and cloudy day.")
```

Conditional statements - example

```
It's a hot and sunny day!
(.venv) ahmadabdellatif@itadmins-MacBook-Pro workspace %
```

What is the output?

```
value = 25

if value > 0:
  if value % 2 == 0:
      print("Positive and even.")
  else:
      print("Positive and odd.")
elif value < 0:
  print("Negative value.")
else:
  print("Zero.")
```

Output:
Positive and odd.

For loop - syntax

Generates a sequence of numbers within a specified range.

The starting value of the sequence/range.

The endpoint of the sequence/range.

Steps is optional and defaults to 1.

```
for variable in range(start, stop, step):
    # Code to be executed in each iteration
```

Example on printing squares of numbers from 1 to 5

```
for num in range(1, 6):

# The double asterisk ** is used as the exponentiation operator
  square = num ** 2
  print(square)
```

Example on printing squares of numbers from 1 to 5

```
1
4
9
16
25
(.venv) ahmadabdellatif@itadmins-MacBook-Pro workspace %
```

Calculating factorial of a number

```python
num = 6
factorial_result = 1

print(f"Calculating factorial of {num}:")

for i in range(1, num + 1):
  factorial_result *= i
  print(f"Factorial after {i} iterations: {factorial_result}")

print(f"The factorial of {num} is: {factorial_result}")
```

Calculating factorial of a number

```
Calculating factorial of 6:
Factorial after 1 iterations: 1
Factorial after 2 iterations: 2
Factorial after 3 iterations: 6
Factorial after 4 iterations: 24
Factorial after 5 iterations: 120
Factorial after 6 iterations: 720
The factorial of 6 is: 720
(.venv) ahmadabdellatif@itadmins-MacBook-Pro workspace %
```

Questions