Lecture II - Control Structures for Your Code

Programming with Python

Dr. Tobias Vlćek

Quick Recap of the last Lecture

F-Strings

- F-strings provide a way to embed expressions inside string literals
- You can include expressions by placing them inside curly braces `{}`
- This makes it easier to include dynamic content

. . .

```
# Let's illustrate f-strings with a small example:
name = "Mr. Smith"
age = 30
height = 1.826549
print(f"My name is {name}, I'm {age} years old, and {height:.2f} meters tall.")
```

```
My name is Mr. Smith, I'm 30 years old, and 1.83 meters tall.
```

. . .

> **Tip**
>
> We used the `:.2f` format specifier to round the number to two decimal places (new).

Variables and Data Types

- Python uses dynamic typing, i.e. the type is determined at runtime
- Basic data types in Python are: `int`, `float`, `str`, `bool`
- Variables are created by assignment with the `=` operator

```
# Let's illustrate these concepts with a small example:
x = 1
y = 2.5
```

```
z = "Hello"
w = True
print(f"x is of type {type(x).__name__}")
print(f"y is of type {type(y).__name__}")
print(f"z is of type {type(z).__name__}")
print(f"w is of type {type(w).__name__}")
```

. . .

> What are the types of `x`, `y`, `z`, `w`?

Arithmetic Operators

Addition
Subtraction
Multiplication
Division
Floor Division
Exponentiation
Modulo

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

Adds two numbers
Subtracts one number from another
Multiplies two numbers
Floating-point division
Integer division
Power of
Remainder of division

. . .

> **Note**
>
> Note, that the `/` operator always returns a float, even if the division is even. Furthermore, the `+` operator can be used to concatenate strings and that the `*` operator can be used to repeat strings.

Arithmetic Operators with Variables

- Additional operators can update the value of a variable (new)
- We can use `+=`, `-=`, `*=`, `/=`, `//=`, `**=`, `%=`

```
x = 10
print(f"Initial value of x: {x}")
x += 5  # Equivalent to x = x + 5
print(f"After x += 5: {x}")
x *= 2  # Equivalent to x = x * 2
print(f"After x *= 2: {x}")
x %= 4  # Equivalent to x = x % 4
print(f"After x %= 4: {x}")
```

. . .

> What is the value of `x` after the operations?

Objects and Methods

Objects

- Objects are **instances of classes**
- We will learn more about classes **later** in the course
- Common built-in objects: integers, strings, lists, dictionaries
- For now, think of objects as a collection of data and methods

Methods

- Methods are functions that are called on an object
- Methods are used to perform operations on an object
- Methods are used to access and modify the attributes of an object
- Methods are used to access and modify the methods of an object
- Methods are used to access and modify the attributes of an object
- Methods are used to access and modify the methods of an object

Indexing and Slicing

Indexing

- We have used indexing to access elements of a string last lecture
- It allows you to access **elements of a sequence** by position
- **Positive indexing** starts at 0 for the first element
- **Negative indexing** starts at -1 for the last element (new)

. . .

```
string_to_index = "Hello, World!"
print(string_to_index[0])  # Accessing the first character
print(string_to_index[-1]) # Accessing the last character
```

```
H
!
```

Slicing

- Slicing allows you to **extract a portion of a sequence**
- Syntax: `sequence[start:stop:step]`
- `start` is the index of the **first element to include**
- `stop` is the index of the **first element to exclude**
- `step` is the increment between indices (default is 1)
- The result is a **new sequence** containing the extracted elements

. . .

```
string_to_slice = "Hello, World!"
print(string_to_slice[7:12])   # Accessing the last five characters from the start
print(string_to_slice[-6:-1])  # Accessing the last five characters from the end
```

```
World
World
```

Slicing Simplified

- If we omit `start` or `stop`, it will be replaced by the start or end of the sequence, respectively
- If we omit `step`, it will be replaced by 1

. . .

```
string_to_slice = "Hello, World!"
print(string_to_slice[::2])   # Accessing every second character
print(string_to_slice[::-1])  # Accessing the string in reverse
```

```
Hlo ol!
!dlroW ,olleH
```

Comparison Operators

- Comparison operators are used to compare two values
- The result of a comparison is a boolean value (`True` or `False`)
- Common comparison operators: `==`, `!=`, `>`, `<`, `>=`, `<=`

Logical Operators

- Logical operators are used to combine multiple comparison operators
- Common logical operators: `and`, `or`, `not`

Membership Operators

- Membership operators are used to check if a value is present in a sequence
- Common membership operators: `in`, `not in`

Identity Operators

- Identity operators are used to check if two values are the same object
- Common identity operators: `is`, `is not`

Control Structures

- Control structures are used to control the flow of execution in a program
- Common control structures: `if`, `elif`, `else`, `for`, `while`

- `if` statements are used to execute a block of code if a condition is true
- `elif` statements are used to execute a block of code if the previous condition is false and the current condition is true
- `else` statements are used to execute a block of code if the previous conditions are false
- `for` loops are used to iterate over a sequence (e.g., list, tuple, string)
- `while` loops are used to execute a block of code repeatedly until a condition is false

Conditional Statements

- Conditional statements allow you to execute different blocks of code based on whether a condition is true or false
- Common conditional statements: `if`, `elif`, `else`

Loops

- Loops allow you to execute a block of code repeatedly
- Common loops: `for`, `while`