Algorithms and Software Concepts
Basics of Python

Marc-Antoine Weisser

CentraleSupélec

September 19, 2024

The interpreter

What is the Python Interpreter?

- **Python interpreter** is the core of the Python language.
- It takes Python code and executes it directly.
- Commonly referred to as the **REPL** (Read-Eval-Print Loop).
- Provides an interactive environment for immediate feedback.
- Common ways to interact with the interpreter:
  - **Interactive mode**: Directly typing Python commands in a shell.
  - **Script mode**: Running Python code from `.py` files.

How the Python Interpreter Works

Compilation vs. Interpretation

- **Compilation** Translating code into bytecode.
- **Interpretation** Executing bytecode on the Python Virtual Machine (PVM).

Step-by-Step Process

- **Parsing** Analyzing syntax.
- **Compilation** Converting to bytecode.
- **Execution** Running on PVM.

Python in IDEs

- While the interpreter can be used from the command line, most developers use IDEs (Integrated Development Environments).
- Popular Python IDEs:
  - PyCharm
  - Spyder
  - VSCode (with Python extension)
- IDEs offer additional features like syntax highlighting, debugging tools, and code suggestions.

For this course, **Thonny** will be used. It is basic and includes all needed features.

Live demonstration of the basic concepts 1

- Basic Arithmetic Operations
  - Variable Assignment
  - Assignment is not a mathematical equality
  - Multi-assignement
- Data type
  - `int`, `float`, `bool`
  - Arithmetic and Logical Operations,
  - Comparison Operators
  - Type Conversion
  - Determining a Type

Live demonstration of the basic concepts 2

- A more complex type `str`
  - Single quote notation
  - Triple quote notation
  - `input` function
- Type of an expression
  - What is an expression
  - expression of `None` type

Scripts

What is a Python Script?

- A Python script is a file containing Python code.
- Usually saved with a `.py` extension.
- Scripts are used for:
  - Automating tasks.
  - Running complex programs.
  - Organizing reusable code.
  - Unlike interactive mode, scripts are executed in one go.

Live demonstration of running a script

- A simple script
- Running with the terminal
- Running with the IDE

Functions

What is a Function?

- A function is a reusable block of code designed to perform a specific task.
- Key Features
  - Encapsulation: Groups related statements together.
  - Reusability: Code can be used multiple times without rewriting.
  - Modularity: Breaks down complex problems into simpler parts.

Defining a function

```
def square(x):
    r = x**x
    return r
```

- `def` function definition
- `square` name of the function
- `(x)` parameters
- `:` beginning of the code block followed with indented code
- `return` value of the function

Calling a function

Actions performed by the interpreter when calling the function on line `s = square(10)`

```
def square(x):
        r = x*x
        return r

s = square(10)
print(s)
```

1. The function `square` is invoked with `x` set to 10
2. The expression `r = x * x` is evaluated, `r` is set to 100
3. The return statement sends the value 100 back to where the function was called
4. The value 100 is assigned to the variable `s`

Scope of Variables

Local scope

- Variables that are defined within a function
- Only accessible within that function
- Cannot be accessed outside of the function
- These variables are destroyed, and their memory is freed after the execution of the function

```
def cube(x):
    y = x * x * x
    return y

cube(10)
print(y)

NameError: name 'y' is not defined
```

Scope of Variables

Global scope

- Variables defined outside of any function
- They can be accessed from any part of the code
- By default changing the value of a global variable will create a local variable which hide the global variable.
- Use of global variables should be minimized
- Usage of global variables must be limited, often for defining constants

```
PLANCK_CONSTANT = 6.626e-34
SPEED_OF_LIGHT = 3.0e8

def calculate_energy(frequency):
    return PLANCK_CONSTANT * frequency
def calculate_time(distance):
    return distance / SPEED_OF_LIGHT
```

Functions parameters

Parameters

- Any number of parameters or none at all.
- When calling a function, you must specify the exact number of parameters in the parentheses.

```
def discriminant(a, b, c):
    return b * b - 4 * a * c
print(discriminant(1, 2, 1))  # Outputs: 0

print(discriminant)
<function discriminant at 0x10115e660>
```

Return values

Return

- The `return` specify what value should be returned
- The value provided after the `return` statement replaces the function call
- Once the return statement is executed, the function exits
- If a function does not use `return`, it implicitly returns `None`

```
def square(x):
    return x * x
    print("This message will never be displayed")

c = square(10) * 5
```

Control flow

The If Statement

- The if statement allows us to execute a block of code only if a specified condition is True.
- It provides a way to make decisions within our programs.

```
def abs(x):
    if x < 0:
        x = -x
    return x

print(abs(10))     # print 10
print(abs(-10))    # print 10
```

Else, Elif

```
# Nested if statements
x = int(input())
if x < 0:
    print("x is neg.")
else:
    if x > 0:
        print("x is pos.")
    else:
        print("x is zero")
```

```
# Using elif
x = int(input())
if x < 0:
    print("x is neg.")
elif x > 0:
    print("x is pos.")
else:
    print("x is zero")
```

Pass

Indentation

- Indentation is part of the syntax.
- We may need a `pass` statement which does nothing except marking the indentation.
- A colon is always followed by indented instruction.

```
def abs(x):
        if x >= 0:
                pass
        else:
                x = -x
        return x
```

While

The while loop

- Allows for the repeated execution of a block of code as long as a specified condition remains True.
- Particularly useful when the number of iterations required is not known.
- In this code, the while loop continues to increment i as long as the square of (i + 1) is less than or equal to the parameter x. Once the condition is no longer satisfied, the loop exits, and the return value is set.

```
def int_sqrt(x):
        i = 0
        while (i + 1) * (i + 1) <= x:
                i += 1
        return i
```

Infinite loop

Infinite loop

- Infinite loop are one of the must classical bug

```
def log10(x):
        i = 0
        while x != 0:
                x = x // 10
        return i

x = int(input())
print(log10(x))
```

Infinite loop

Infinite loop

- Infinite loop are one of the must classical bug

```
def log10(x):
        if x < 0:
                return None
        i = 0
        while x != 0:
                x = x // 10
                i += 1
        return i

x = int(input())
print(log10(x))
```

continue, break

The `continue` statement returns at the beginning of the while loop.

```
x = 1
while x < 10:
        if x % 2 == 0:
                continue
print(x)
```

The `break` statement immediately terminates the loop.

```
x = 1
while True:
    print(x)
    x += 1
    if x % 5 == 0:
        break

print(".")
```

Demo: Secret Number Game

Creating a game

Demo.

Conclusion

- The program is developed incrementally
- Each step can be run even if it doesn't yet perform all tasks
- Helps avoid accumulating syntax errors
- By developing the program function by function, you can test each part independently and identify bugs more easily