Python-Basics
Introduction

Er. Narayan Sapkota
M.Sc. Computational Science and Engineering

Kathmandu University (Visiting Faculty)

December 5, 2024

Table of Contents

1. Get Your Machine Ready !
   - Installing and Running Python
   - Jupyter Notebook
   - Working with Virtual Environment
2. Python Basics
   - Why Python?
   - Comments
   - Indentation
   - Tokens
   - Identifiers
   - Keywords
   - Statements and Expressions
   - Literals
   - Variables
   - Constants

Installing and Running Python

IDLE

Installing Python Libraries/Packages/Modules (1)

- Module: A file containing Python code (e.g., functions, classes) that can be imported into other Python programs.
- Package: A collection of Python modules grouped together in a directory.

Why Install External Packages?

- Python has a rich ecosystem of libraries for various tasks (e.g., data analysis, web scraping).
- External packages save time and effort by providing pre-written solutions for common tasks.

Using `pip` (Python's Package Installer)

- Install a package: `pip install numpy`
- Install a specific version: `pip install numpy==1.18.5`
- Install from requirements file: `pip install -r requirements.txt`

Installing Python Libraries/Packages/Modules (2)

Verifying Installation

```
import numpy as np
print(np.__version__)
```

Upgrading a Package: `pip install --upgrade numpy`
Uninstalling a Package: `pip uninstall numpy`

Jupyter Notebook

Working with Virtual Environment

Python Introduction (1)

- Python is a high-level, interpreted programming language known for its simplicity and readability.
- Developed by Guido van Rossum in the late 1980s and released in 1991.
- Named after the British comedy group Monty Python, reflecting its fun and approachable design.
- Python emphasizes code readability with a clean, easy-to-understand syntax.
- It is platform-independent, meaning Python code can run on any operating system without modification.
- Python supports multiple programming paradigms:
  - Imperative programming: describes a sequence of instructions to change the program's state.
  - Object-oriented programming: programs are organized around objects that contain both data and methods.

Python Introduction (2)

  - Functional programming: treats computation as the evaluation of mathematical functions and avoids changing state or mutable data.
- Major uses of Python:
  - Web development (e.g., Django, Flask)
  - Data science and machine learning (e.g., Pandas, TensorFlow)
  - Automation, scripting, and system administration
  - Scientific computing (e.g., NumPy, SciPy)
  - Game development, networking, and more.
- Python's syntax is clean and intuitive, making it a great choice for beginners.
- The language has a large, active community, with extensive libraries and frameworks for various applications.

The Zen of Python (1)

The language's core philosophy is summarized in the document The Zen of Python which includes principles such as,

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.

The Zen of Python (2)

- In the face of ambiguity, refuse the temptation to guess.
- There should be one—and preferably only one—obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea—let's do more of those!

Why Python?

- Easy to Learn and Use: Designed to be simple and easy to understand, with clear and readable syntax, making it ideal for beginners.
- Versatile and Flexible: Supports multiple programming paradigms, including procedural, object-oriented, and functional programming.
- Large Community and Resources: Has a vast community of developers, ensuring a wealth of resources, libraries, and frameworks are available to help solve various problems.
- Platform Independent: Code can run on any platform, including Windows, macOS, and Linux, without modification.
- Wide Application: Used in web development, data science, machine learning, artificial intelligence, automation, and many other fields.
- Extensive Libraries: Python's extensive standard library and external packages make it suitable for almost any task, from scientific computing to web development.

input(): Reading Input (1)

- The `input()` function is used to gather data from the user.

```
variable_name = input([prompt])
```

- Prompt: A string inside the parentheses displayed on the screen to guide the user in providing input.
- When the user presses Enter, the program continues, and `input()` returns the input as a string.

```
>>> person = input("What is your name?")
  What is your name? Narayan Sapkota
>>> person
  'Narayan Sapkota'
```

- The input is assigned as a string to the variable `person`.
- The input is captured and stored as a string, even if it's a number. Therefore, explicit conversion to numeric types is required.

input(): Reading Input (2)

```
>>> number = input("Enter a number: ")
  Enter a number: 42
>>> number
  '42'
```

- To convert it to a numeric type, explicit casting is needed:

```
>>> number = int(input("Enter a number: "))
  Enter a number: 42
>>> number
  42
```

Here, the input is converted from a string to an integer using `int()`.

print(): Output Function and String Formatting

- The print() function displays output on the console. It converts any non-string input into its string representation automatically.

```
>>> print("Hello World!!")
  Hello World!!
```

- Python offers two common string formatting methods:
  - str.format()
  - f-strings (introduced in Python 3.6)

str.format() Method (1)

- This method allows inserting variables or expressions into a string.

```
str.format(p0, p1, ..., k0=v0, k1=v1, ...)
```

```
country = input("Which country do you live in?")
print("I live in {0}".format(country))
```

Output:

```
Which country do you live in?  India
I live in India
```

- You can also change the order of positional arguments:

```
a = 10
b = 20
print("The values of a is {0} and b is {1}".format(a, b)
    )
print("The values of b is {1} and a is {0}".format(a, b)
    )
```

str.format() Method (2)

Output:

```
The values of a is 10 and b is 20
The values of b is 20 and a is 10
```

- Example using keyword arguments:

```
print("Give me {ball} ball".format(ball="tennis"))
```

Output:

```
Give me tennis ball
```

f-strings (1)

- f-strings provide an easier and more readable way to format strings in Python 3.6 and later.

```
print(f"Some text variable")
```

Program 1: f-string example

```
country = input("Which country do you live in?")
print(f"I live in {country}")
```

Output:

```
Which country do you live in?  India
I live in India
```

f-strings (2)

Program 2: Area and Circumference of a Circle

```
radius = int(input("Enter the radius of a circle:"))
area = 3.1415 * radius * radius
circumference = 2 * 3.1415 * radius
print(f"Area = {area} and Circumference = {circumference
    }")
```

Output:

```
Enter the radius of a circle:  5
Area = 78.53750000000001 and Circumference = 31.415000000000003
```

Comments

- Comments are explanatory notes added to the code to make it easier to understand for anyone reading, ignored by the interpreter. They're there purely for human benefit– understanding, maintaining, and debugging code.
- Single-line Comment: Use # to comment a single line.

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

- Multiline Comment: Use # at the beginning of each line. Alternatively, use triple quotes (''' or """) for multiline comments.

```
# This is a multiline
# comment example.
print("Multiline comment")
"""This is just
a another style
of commenting in python. """
```

Indentation

- Python uses indentation (whitespace) to define code blocks instead of curly braces ({}).
- The standard indentation is 4 spaces per level.
- Improper indentation will result in an `IndentationError`.
- Indentation makes code more readable and structured.

Program 3: Indentation example

```
if True:
    print("This is indented correctly.")  # Correct
        indentation
if False:
    print("This will not print.")  # Nested block
    print("End of block.")  # Back to the outer block
```

Tokens

A token is the smallest unit of a program that has a meaning in Python. Tokens are the building blocks of a program, used by the interpreter to understand the program. Types of tokens in Python are:

- Keywords: Reserved words with special meaning (e.g., `if`, `for`, `while`).
- Identifiers: Names for variables, functions, classes, etc. (e.g., `a`, `my_fxn`).
- Literals: Constants or values used in the program (e.g., `10`, `"Hello"`).
- Operators: Symbols that perform operations (e.g., `+`, `-`, `*`).
- Delimiters: Symbols used to separate statements or define code blocks (e.g., `()` for grouping, `:` for defining blocks).

```
my_val = 10              # 'my_val' is an identifier, '10' is a
    literal, '=' is an operator
if my_val > 5:           # 'if' is a keyword, 'my_val' is an
    identifier, '>' is an operator
print("Greater than 5") # 'print' is a function call (
    identifier), "Greater than 5" is a literal
```

Identifiers

- An identifier is a name given to a variable, function, class, or module.
- Identifiers can be a combination of letters (lowercase `a to z` and uppercase `A to Z`), digits `0 to 9`, and underscores `_`.
- Valid examples of identifiers include: `myCount`, `other_1`, `good_morning`
- Identifiers may be one or more characters in the following format:
  - An identifier must begin with a letter `A-Z` or `a-z`, or an underscore `_`.
  - An identifier cannot start with a digit. `1plus` is invalid, but `plus1` is valid.
  - Keywords cannot be used as identifiers.`def` or `class` are reserved and cannot be used as identifiers.
  - Identifiers cannot contain spaces or special symbols, such as `,` `.` `#`, `$`, `%`, `etc.`.
  - An identifier can be of any length.

Keywords

- Keywords are reserved words that have predefined meanings and cannot be used as identifiers.
- These words are part of the syntax of the language, and they dictate the structure and flow of a program. Python has a set of keywords that are used for various purposes like control flow, functions, and classes.

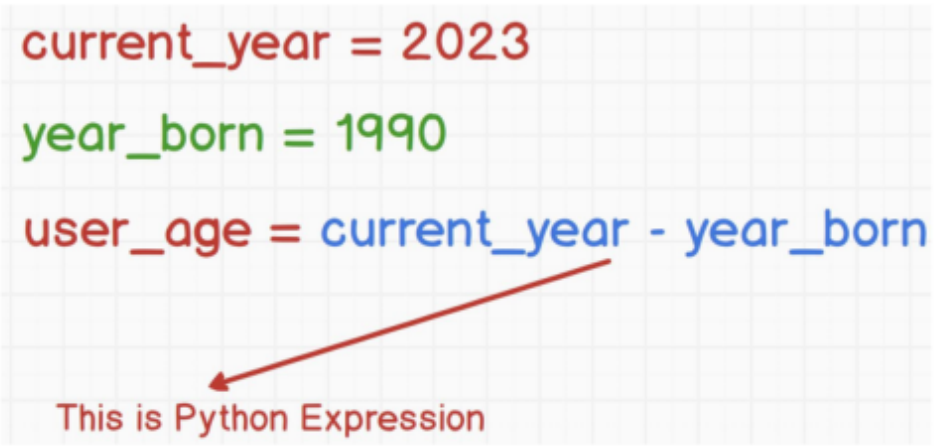
```
import keyword
print(keyword.kwlist)

['False', 'None', 'True', 'and', 'as', 'assert', 'async'
    , 'await', 'break', 'class', 'continue', 'def', 'del
    ', 'elif', 'else', 'except', 'finally', 'for', 'from
    ', 'global', 'if', 'import', 'in', 'is', 'lambda', '
    nonlocal', 'not', 'or', 'pass', 'raise', 'return', '
    try', 'while', 'with', 'yield']
```

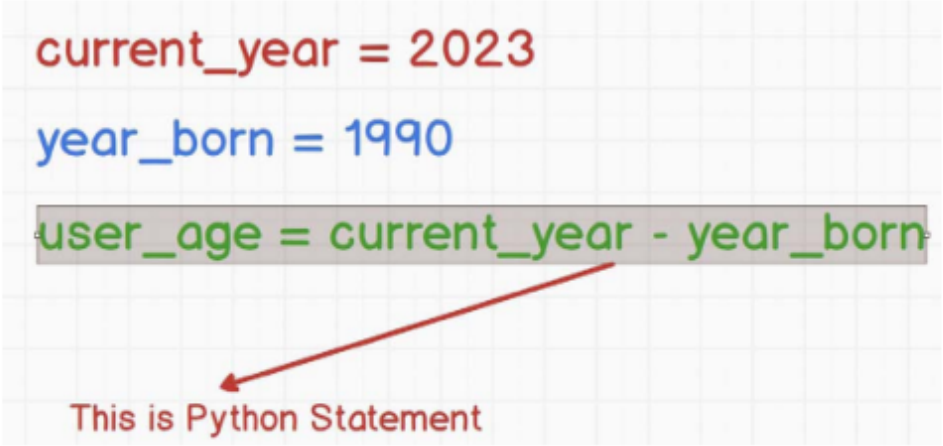
- Keywords are case-sensitive, meaning `True` is a keyword, but `true` is not.

Statements and Expressions (1)

- Statement: A statement is an instruction that the Python interpreter can execute. It performs an action but does not return a value.
- Common examples of statements:
  - Assignment statement: `a = 5`
  - Control flow statements: `if`, `while`, `for`
  - Function definition: `def my_function():`
- Expression: An expression is a combination of values, variables, operators, and function calls that produces a value when evaluated.
- Examples of expressions are:
  - `5 + 10` (Evaluates to 15)
  - `a * b` (Evaluates to the product of `a` and `b`)
  - `len("Hello")` (Evaluates to 5, the length of the string "Hello")

Statements and Expressions (2)





- Key Difference: A statement performs an action but does not produce a value, while an expression produces a value when evaluated.

Program 4: statement vs expression

```
# Statement
a = 10

# Expression
result = a + 5    # Evaluates to 15
```

Literals

- A literal is a fixed value used directly in a program. It represents data that is assigned to variables or used in expressions.
- Literals are values that you write in the code, which represent constant data. They can be of several types:
- String Literals: Represented by sequences of characters enclosed in quotes. Examples: `"Hello, World!"`, `'Python'`
- Numeric Literals: Represent numbers. They can be integers, floating-point numbers, or complex numbers.
  - Integer literal: `42`, `-7`, `0`
  - Float literal: `3.14`, `-2.0`
  - Complex literal: `1+2j`
- Boolean Literals: Represent the Boolean values `True` or `False`.
- Special Literals: Represent the special value `None`, used to represent the absence of a value or a null value. Examples: `None`, Ellipsis (Represented by `...`, used as a placeholder or to indicate that a code block is incomplete or missing.)

Variables (1)

- A named placeholder that holds data used by the program. In Python, you don't need to declare the type explicitly.
- Legal Variable Names:
  - Can consist of letters, digits, and underscores.
  - Cannot start with a digit.
  - Cannot be Python keywords.
  - Case-sensitive (e.g., `computer` & `Computer` are different).
  - Use `lowercase_letters` with underscores for readability (e.g., `whats_up`).
  - Avoid starting with an underscore unless necessary.
- Assigning Values to Variables:

```
number = 100          # Integer
miles = 1000.0        # Float
name = "Python"       # String
```

- Variables can change their values and types during execution.

Variables (2)

```
century = 100
print(century)          # Output: 100
century = "hundred"
print(century)          # Output: 'hundred'
```

- Simultaneous Assignment:
  - Python allows assigning the same value to multiple variables at once.

```
a = b = c = 1
print(a, b, c)   # Output: 1 1 1
```

Constants

- A constant is a value that does not change during the execution of the program. Python does not have built-in support for constants, but you can use naming conventions to represent constants.
- Convention: By convention, constants are defined using UPPERCASE letters with words separated by underscores (e.g., MAX_VALUE, PI).
- Although Python does not prevent you from changing the value of a constant, the naming convention serves as a signal that the value should not be changed.

```
PI = 3.14159              # Constant value for Pi
MAX_VALUE = 1000          # Maximum limit value
```

- Why Use Constants?
  - Constants help improve readability and maintainability by giving meaningful names to fixed values.
  - Using constants reduces the chance of accidentally changing important values in the program.