BUREAUTIQUE

Python Programming

Prepared by:

Nassim KADDOURI

Academic Year: 2024-2025

NUMIDIA INSTITUTE OF TECHNOLOGY

Chapter 1

Introduction to Python

What is Python?

Python is a popular programming language known for its simplicity and readability. It is used for various tasks, from web development and data analysis to artificial intelligence and automation. Python is a great language for beginners because its **syntax** (the rules for writing code) is easy to learn and understand.

Here is a simple example of a Python program:

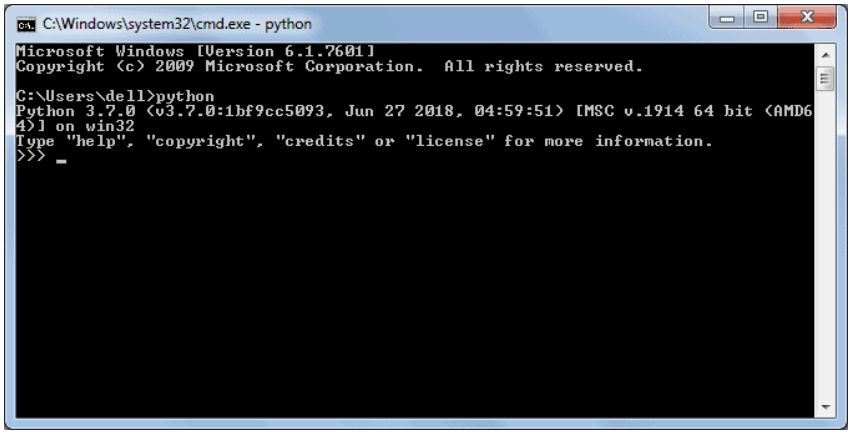
```
# A simple program to greet the user
name = input("Enter your name: ")
print("Hello, " + name + "!")
```

In this example, the program asks for your name and then greets you.

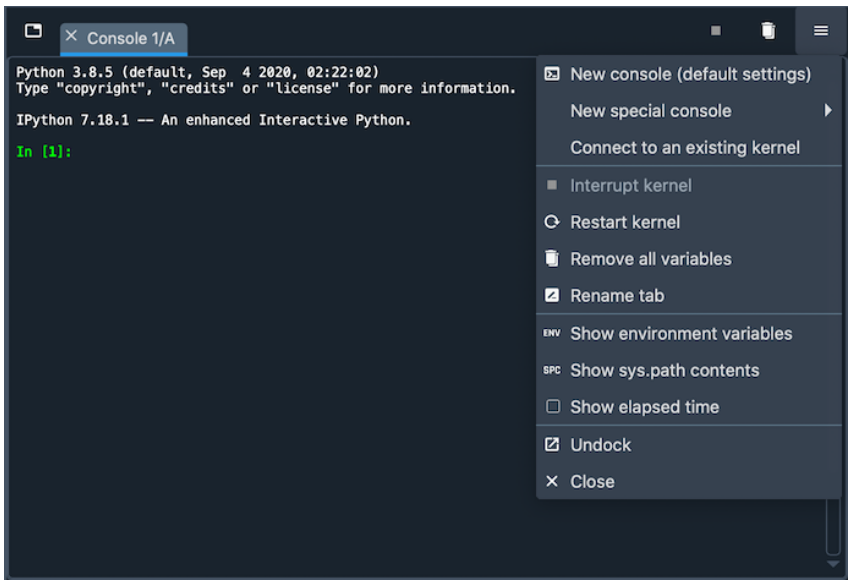
What is a Console?

The console, also known as the command line or terminal, is a text-based interface used to run programs and display output.

Here is an example of what a console might look like on windows:



Here is an example of what a console might look like on Spyder:



When you write a Python program, you can run it in the console to see the results.

1.1 My First Python Code!

- **Example:**

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

- **Output:**

```
Hello, World!
```

- **Description:**
  - The ritual for programmers when they start coding with a new language is to output **Hello, World!** on the **console**.

1.2 Comments in Python

- **Syntax:**

```
# This is a comment
```

- **Description:**
  - Comments start with a `#` and are ignored by the **Python interpreter** and automatically goes to the next line.
  - Comments are used to organize our work and allow it to be understood by readers.
  - Comments don't output anything

1.3 Outputs

- **Syntax:**

```
print(value)
```

- **Description:**
  - The `print()` function is used to output a value to the console.
- **Example usage:**

```
print("Hello")
print(42)
```

- **Output:**

```
Hello, World!
42
```

4. Variables and Variable Types

- **Syntax:**
  - For `int` (whole number) value:

```
x = 5
```

– For `str` (string, text) value:

```
#We can use both " " or ' ' for str
name = "Hamoud"
surname = 'Bou3lam'
```

– For `float` (decimal number) value:

```
#We use a . and not a ,
pi = 3.14
```

- **Description:**
  - Variables are used to store data.
  - When naming a variable we cannot start its name with a number or a special character (apart from _).
  - As opposed to C, where variable types must be explicitly declared (e.g., `int`, `float`, `char`), Python does not require type declaration, making it more flexible.
- **Example usage:**

```
x = 42                  # Integer
print(x)

pi = 3.14159            # Float
print(pi)

name = "Bob"            # String
print(name)

is_student = False      # Boolean
print(is_student)
```

- **Output:**

```
42
3.14159
Bob
False
```

- **Important Note:**
  - To print the type of a **value/variable** we use the function `type()`:
    Syntax:

```
print(type(value))
```

Example Usage:

```
x = 5
#print type of a variable
print(type(x))
#print type of a value
print(type(1.44))
```

Output:

```
<class 'int'>
<class 'float'>
```

5. Operations on Values and Variables

- **Arithmetic Operations:**
  - **Addition:** `x + y`

```
result = x + y
print(result)
```

  - **Subtraction:** `x - y`

```
result = x - y
print(result)
```

  - **Multiplication:** `x * y`

```
result = x * y
print(result)
```

  - **Division:** `x / y` (always returns float)

```
result = x / y
print(result)
```

– **Euclidean Division:** `x // y` (returns an integer)

```
result = x // y
print(result)
```

– **Modulus (remainder):** `x % y`

```
result = x % y
print(result)
```

– **Power:** `x ** y`

```
result = x ** y
print(result)
```

- **Example Usage:**

```
x = 10
y = 3
print(x + y)  # Addition
print(x - y)  # Subtraction
print(x * y)  # Multiplication
print(x / y)  # Division
print(x // y) # Floor Division
print(x % y)  # Modulus
print(x ** y) # Exponentiation
```

- **Output:**

```
13
7
30
3.3333333333333335
3
1
1000
```

6. Input

- **Syntax:**

```
x = input("Ask user to input something: ")
```

Or alternatively:

```
print("Ask user to input something: ")
x = input()
```

- **Description:**
  - The `input()` function reads user input from the console. It displays a message and waits for the user to type something and press Enter. The input is always stored as a string.
- **Example usage:**

```
name = input("Enter your name: ")
print("Hello, " + name)
```

- **Output:**

```
Enter your name: Alice
Hello, Alice
```

- **Important Note:**
  - The value read using the `input()` function is always a string. To use it as a different data type, such as an integer, you need to convert it using type casting.
  - If you don't convert your value, you won't be able to use the arithmetic operations such as +.
  - **Example:**

```
x = input("Enter a number: ")
y = input("Enter another number: ")
result = x + y
print(result)
# or alternatively
# print(x+y)
```

– **Output:**

```
Enter a number: 5
Enter another number: 3
53
```

– To cast a string to an integer, use the `int()` function.

– To cast a string to a float, use the `float()` function.

– **Example Usage:**

```
x = int(input("Enter a number: "))
y = int(input("Enter another number: "))
# or alternatively
# x = input("Enter a number: ")
# y = input("Enter another number: ")
# x = int(x)
# y = int(y)
result = x + y
print(result)
```

– **Output:**

```
Enter a number: 5
Enter another number: 3
8
```

Chapter 2

Conditions and Loops in Python

1. Conditions

- **Description:**
  - Conditions allow your program to make decisions based on whether certain statements are true or false.
  - In Python, we use the keywords `if`, `elif`, and `else` to express conditions.
- **Syntax:**

```
if condition1:
    # code to execute when condition is true
elif condition2:
    # code to execute if another_condition is
      true
else:
    # code to execute if no condition is true
```

- **Comparison Operators:**
  - `==` checks if two values are equal.
  - `!=` checks if two values are not equal.
  - `>` checks if the left value is greater than the right.
  - `<` checks if the left value is less than the right.
  - `>=` checks if the left value is greater than or equal to the right.
  - `<=` checks if the left value is less than or equal to the right.

- **Example Usage:**

```
age = int(input("Enter your age: "))

if age >= 18:
    print("You are an adult.")
elif age > 12:
    print("You are a teenager.")
else:
    # age < 12
    print("You are a child.")
```

- **Output:**

```
Enter your age: 15
You are a teenager.
```

- **Logical Operators:**
  Logical operators are used to combine multiple conditions in a single expression. The three primary logical operators in Python are:
  - `and` - Returns `True` only if **both conditions** are true.
  - `or` - Returns `True` if **at least one condition** is true.
  - `not` - Reverses the result of a condition, returning `True` if the condition is false.

```
if condition1 and condition2:
    # code to execute when both conditions are
      true
if condition1 or condition2:
    # code to execute if one of the conditions
      is true
if not condition1:
    # code to execute if condition1 is false
```

- **Example Using Logical Operators:**

```
print("Enter the temperature in degrees: ")
temperature = int(input())

if temperature > 30 and temperature < 40:
    print("It's warm outside.")
elif temperature >= 40:
    print("It's very hot outside!")
else:
    print("The temperature is moderate.")
```

- **Output:**

```
Enter the temperature in degrees: 35
It's warm outside.
```

2. For Loops

- **Description:**
  - A **for loop** is used to iterate over a sequence (such as a list, tuple, string, or range of numbers).
  - It executes a block of code for each item in the sequence.
- **Syntax:**

```
for item in sequence:
    # code to execute for each item
```

- **Example Usage:**

```
for i in range(5):
    print("Iteration:", i)
```

- **Output:**

```
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
```

- **Important Note:**
  - The `range()` function generates a sequence of numbers. By default, it starts at 0 and increments by 1.
  - Syntax of `range()`:
    - `range(stop)`: Generates numbers from 0 to `stop - 1`.
    - `range(start, stop)`: Generates numbers from `start` to `stop - 1`.
    - `range(start, stop, step)`: Generates numbers from `start` to `stop - 1`, incrementing by `step`.
- **Example with Step:**

```
for i in range(1, 10, 2):
    print("Odd number:", i)
```

- **Output:**

```
Odd number: 1
Odd number: 3
Odd number: 5
Odd number: 7
Odd number: 9
```

3. While Loops

- **Description:**
  - A **while loop** continues to execute a block of code as long as the specified condition is `True`.
- **Syntax:**

```
while condition:
    # code to execute while condition is true
```

- **Example Usage:**

```
count = 0
while count < 5:
    print("Count:", count)
    count += 1
```

- **Output:**

```
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
```

- **Important Note:**
  - Be cautious of infinite loops, which occur when the condition never becomes `False` or when we use True.

```
while True:
    #code to run for infinity
```

4. Break Statement

- **Description:**
  - The **break** statement is used to exit a loop prematurely when a certain condition is met.
- **Syntax:**

```
for item in sequence:
    if condition:
        break
    # code to execute if no break

while condition:
    if another_condition:
        break
    # code to execute if no break
```

- **Example Usage in For Loop:**

```
for i in range(10):
    if i == 5:
        print("Breaking the loop at:", i)
        break
    print(i)
```

- **Output:**

```
0
1
2
3
4
Breaking the loop at: 5
```

- **Example Usage in While Loop:**

```
count = 0
while count < 10:
    if count == 7:
        print("Breaking the loop at:", count)
        break
    print(count)
    count += 1
```

- **Output:**

```
0
1
2
3
4
5
6
Breaking the loop at: 7
```

Chapter 3

Python Data Structures

1. Python Lists

- **Description:**
  - A **list** is a collection that is ordered, mutable, and allows duplicate elements.
  - Lists are defined using square brackets: `[]`.
- **Syntax:**

```
list_name = [item1, item2, item3]
```

- **Example:**

```
my_list = [1, 2, 3, "apple", "banana"]
print(my_list)
```

- **Output:**

```
[1, 2, 3, 'apple', 'banana']
```

- **Accessing Elements:**

```
my_list = [10, 20, 30, 40]
print(my_list[0])  # First element
print(my_list[-1]) # Last element
```

- **Output:**

```
10
40
```

2. Common List Operations

- **Adding Elements:**

```
# Append an element
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)

# Insert an element
my_list.insert(1, "inserted")
print(my_list)
```

- **Output:**

```
[1, 2, 3, 4]
[1, 'inserted', 2, 3, 4]
```

- **Removing Elements:**

```
my_list = [1, 2, 3, 4]
my_list.remove(2)  # Remove specific element
print(my_list)

# Remove and return last element
popped = my_list.pop()
print(my_list)
print("Popped element:", popped)
```

- **Output:**

```
[1, 3, 4]
[1, 3]
Popped element: 4
```

- **Slicing:**

```
my_list = [0, 1, 2, 3, 4, 5]
print(my_list[1:4])  # from index 1 to 3
print(my_list[:3])   # from start to index 2
print(my_list[3:])   # from index 3 to end
```

- **Output:**

```
[1, 2, 3]
[0, 1, 2]
[3, 4, 5]
```

- **Other Operations:**

```
my_list = [3, 1, 4, 2]
print(len(my_list))       # Length of the list
print(sorted(my_list))    # Sorted list
my_list.reverse()         # Reverse the list
print(my_list)
```

- **Output:**

```
4
[1, 2, 3, 4]
[2, 4, 1, 3]
```

3. Strings in Python

- **Description:**
  - A **string** is a sequence of characters enclosed in quotes.
  - Strings are immutable, meaning their contents cannot be changed after creation.
- **Syntax:**

```
string_name = "This is a string"
```

- **Accessing Characters:**

```
my_string = "Hello"
print(my_string[0])  # First character
print(my_string[-1]) # Last character
```

- **Output:**

```
H
o
```

4. Common String Operations

- **Concatenation and Repetition:**

```
str1 = "Hello"
str2 = "World"
print(str1 + " " + str2)  # Concatenate strings
print(str1 * 3)           # Repeat string
```

- **Output:**

```
Hello World
HelloHelloHello
```

- **String Methods:**

```
my_string = "Python Programming"
print(my_string.lower())    # Convert to
  lowercase
```

```
print(my_string.upper())     # Convert to
  uppercase
print(my_string.replace("Python", "Java")) #
  Replace substring
```

- **Output:**

```
python programming
PYTHON PROGRAMMING
Java Programming
```

- **Splitting and Joining:**

```
my_string = "apple,banana,cherry"
fruits = my_string.split(",")  # Split by comma
print(fruits)
# Join list into string
joined = " ".join(fruits)
print(joined)
```

- **Output:**

```
['apple', 'banana', 'cherry']
apple banana cherry
```

- **String Formatting:**

```
name = "Dja3fer"
age = 18
print(f"My name is {name} and I am {age}.")  #
  f-string
```

- **Output:**

```
My name is Dja3fer and I am 18.
```

5. Combining Lists and Strings

- **Example:**

```
names = ["Alice", "Bob", "Charlie"]
greetings = [f"Hello, {name}!" for name in
  names]
print(greetings)
```

- **Output:**

```
['Hello, Alice!', 'Hello, Bob!', 'Hello,
  Charlie!']
```