Programming with Python

This comprehensive guide will cover various fundamental topics in Python programming. It aims to provide detailed explanations, examples, and code snippets to help you understand and master these concepts. The topics covered in this guide include:

Let's get started!

Variables and Arithmetic

Variables

In Python, variables are used to store data. They act as containers that hold values of different types, such as numbers, strings, or more complex data structures. To assign a value to a variable, you can use the assignment operator `=`. Here's an example:

```
x = 10
y = "Hello, World!"
```

Arithmetic Operations

Python supports various arithmetic operations, including addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), and exponentiation (`**`). Here are a few examples:

```
a = 5
b = 2

sum = a + b          # Addition
difference = a - b   # Subtraction
product = a * b      # Multiplication
quotient = a / b     # Division
exponent = a ** b    # Exponentiation
```

Functions and Getting Help

Functions

Functions in Python allow you to encapsulate reusable blocks of code. They take input arguments (if any) and can return a value (or not). You can define your own functions using the `def` keyword. Here's an example of a function that calculates the sum of two numbers:

```
def add_numbers(a, b):
    sum = a + b
    return sum
```

You can call this function by passing two numbers as arguments:

```
result = add_numbers(3, 4)
print(result)  # Output: 7
```

Getting Help

To get help on any Python object or function, you can use the `help()` function or access the documentation using the `__doc__` attribute. For example:

```
help(print)           # Get help for the print() function
print(print.__doc__)  # Access the documentation for the print() function
```

Data Types

Numeric Types

Python supports several numeric types, including integers (`int`), floating-point numbers (`float`), and complex numbers (`complex`). Here's an example:

```
# Integer
x = 10

# Floating-point number
y = 3.14

# Complex number
z = 2 + 3j
```

Strings

Strings are used to represent text data in Python. You can define strings using either single quotes (') or double quotes (").

```
message = "Hello, World!"
```

You can perform various operations on strings, such as concatenation (`+`), slicing, and accessing individual characters.

Booleans

Booleans represent the truth values `True` or `False`. They are often used in conditions and logical operations. Here's an example:

```
is_valid = True
is_invalid = False
```

Lists

Lists are ordered collections of items, which can be of different types. You can create a list by enclosing comma-separated values in square brackets (`[]`). Here's an example:

```
fruits = ["apple", "banana", "cherry"]
```

You can access individual items in a list using their index and perform various operations, such as adding or removing items.

Sets

Sets are unordered collections of unique elements. You can create a set by enclosing comma-separated values in curly braces (`{}`). Here's an example:

```
colors = {"red", "green", "blue"}
```

Sets support mathematical set operations like union, intersection, and difference.

Dictionaries

Dictionaries are unordered collections of key-value pairs. Each value is associated with a unique key, allowing efficient retrieval of data. You can create a dictionary by enclosing key-value pairs in curly braces (`{}`). Here's an example:

```
person = {
    "name": "John",
    "age": 30,
    "city": "New York"
}
```

You can access values by their keys and perform various operations, such as adding or removing key-value pairs.

Conditions and Conditional Statements

Conditional Operators

Python provides various conditional operators to compare values. These include:

- Equal to (==)
- Not equal to (!=)
- Greater than (>)
- Less than (<)
- Greater than or equal to (>=)
- Less than or equal to (<=)

If Statements

If statements allow you to execute different blocks of code based on certain conditions. Here's an example:

```
age = 18

if age >= 18:
    print("You are an adult.")
else:
    print("You are not an adult.")
```

You can also use `elif` to specify additional conditions.

Lists, Sets, and Dictionaries

Lists

Lists are mutable, ordered collections that can store elements of different types. Here are some common operations you can perform on lists:

```
fruits = ["apple", "banana", "cherry"]

# Accessing elements
print(fruits[0])    # Output: apple

# Modifying elements
fruits[1] = "pear"
print(fruits)       # Output: ['apple', 'pear', 'cherry']

# Adding elements
fruits.append("orange")
```

```
print(fruits)       # Output: ['apple', 'pear', 'cherry', 'orange']

# Removing elements
fruits.remove("apple")
print(fruits)       # Output: ['pear', 'cherry', 'orange']
```

Sets

Sets are mutable, unordered collections of unique elements. Here are some common operations you can perform on sets:

```
colors = {"red", "green", "blue"}

# Adding elements
colors.add("yellow")
print(colors)       # Output: {'red', 'green', 'blue', 'yellow'}

# Removing elements
colors.remove("red")
print(colors)       # Output: {'green', 'blue', 'yellow'}
```

Dictionaries

Dictionaries are mutable, unordered collections of key-value pairs. Here are some common operations you can perform on dictionaries:

```
person = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

# Accessing values
print(person["name"])   # Output: John

# Modifying values
person["age"] = 31
print(person)           # Output: {'name': 'John', 'age': 31, 'city': 'New
York'}

# Adding new key-value pairs
person["gender"] = "Male"
print(person)           # Output: {'name':

 'John', 'age': 31, 'city': 'New York', 'gender': 'Male'}

# Removing key-value pairs
del person["city"]
```

```
print(person)           # Output: {'name': 'John', 'age': 31, 'gender':
'Male'}
```

Loops: while and for, continue, break

while Loop

The `while` loop repeatedly executes a block of code as long as a given condition is true. Here's an example:

```
count = 0

while count < 5:
    print(count)
    count += 1
```

for Loop

The `for` loop iterates over a sequence of elements. Here's an example:

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print(fruit)
```

continue and break Statements

- The `continue` statement is used to skip the current iteration and move to the next one in the loop.
- The `break` statement is used to exit the loop prematurely. It terminates the loop and continues with the next statement after the loop.

```
for i in range(10):
    if i % 2 == 0:
        continue    # Skip even numbers
    elif i == 7:
        break       # Exit the loop when i is 7
    print(i)
```

Exceptions

Handling Exceptions

Exceptions are errors that occur during the execution of a program. You can handle exceptions using the `try-except` block. Here's an example:

```
```

```
try:
    result = 10 / 0
    print(result)
except ZeroDivisionError:
    print("Error: Division by zero occurred.")
```

Raising Exceptions

You can raise exceptions manually using the `raise` statement. Here's an example:

```
age = -5

if age < 0:
    raise ValueError("Age cannot be negative.")
```

External Libraries

Installing External Libraries

To use external libraries in Python, you need to install them. The most common way to install libraries is using the package manager `pip`. For example, to install the `requests` library, you can run the following command in your terminal:

```
pip install requests
```

Importing External Libraries

Once a library is installed, you can import it into your Python code using the `import` statement. Here's an example:

```
import requests

response = requests.get("https://www.example.com")
print(response.status_code)
```

Basics of Terminal Commands

Running Python Files

To run a Python file, open your terminal or command prompt and navigate to the directory where the file is located. Then, use the `python` command followed by the file name and its extension. For example:

```
python my_script.py
```

Navigating Directories

- `cd directory_name` - Change to a specific directory.
- `cd ..` - Move up to the parent directory.
- `cd /` - Move to the root directory.

Listing Files and Directories

- `ls` - List files and directories in the current directory.
- `ls -l` - List files and directories with detailed information.
- `ls -a` - List all files and directories, including hidden ones.

Creating and Deleting Files/Directories

- `touch file_name` - Create an empty file.
- `mkdir directory_name` - Create a new directory.
- `rm file_name` - Delete a file.
- `rmdir directory_name` - Delete an empty directory.

This Python programming guide covers various fundamental topics to get you started. Experiment with the provided examples and continue exploring Python's vast capabilities. Happy coding!