Notebook For Learning Python Programming Language

Module 1

Introduction to Python

Python is a high-level, interpreted programming language that is widely used in software development, data analysis, artificial intelligence, and many other fields. It is known for its simplicity, readability, and flexibility, making it a great language for beginners as well as experienced developers.

Setting up a development environment

There are several ways to set up a development environment for Python, depending on your operating system and preferences. Here are a few common methods:

1. Using the Python Standard Library: Python comes with a built-in development environment called IDLE. You can use IDLE to write, run, and debug Python scripts. To access IDLE, simply open your terminal and type "python" or "python3" (depending on your installation) to open the interactive interpreter.

2. Using Anaconda: Anaconda is a popular distribution of Python that comes with many commonly used libraries and tools for data science and scientific computing. It also includes a built-in development environment called Anaconda Navigator, which provides a user-friendly interface for managing Python environments and packages. You can download Anaconda from the official website.

3. Using Virtual Environments: Virtual environments allow you to create isolated Python environments for different projects, so that each project can have its own dependencies and versions of Python packages. You can use the venv module in the Python standard library to create virtual environments, or you can use a third-party tool such as virtualenv or conda.

4. Using a Text Editor or IDE: You can use a text editor, such as Sublime Text or Atom, to write Python code, and then run the scripts from the command line. Alternatively, you can use an integrated development environment (IDE), such as PyCharm, Spyder, or VSCode, which provide more advanced features such as code completion, debugging, and integration with version control systems.

Basic syntax and Data types

In Python, data types are used to define the type of a variable, and the type of data that a variable can store. Python has several built-in data types, including:

1. Numbers: Python supports several types of numbers, including integers (int), floating-point numbers (float), and complex numbers (complex).

In [ ]:
```python
# Examples of numbers
a = 5 # integer
```

```
b = 5.5 # float
c = 5 + 6j # complex
```

2. Strings: Strings are sequences of characters. They are used to represent text and can be defined using either single or double quotes. Strings can be concatenated and repeated, and individual characters can be accessed using indexing.

In [ ]:
```
s = "Hello, World!"
print(s)
print(s[0]) # "H"
print(s[1:5]) # "ello"
print(s * 2) # "Hello, World!Hello, World!"
print(s + " " + "Python") # "Hello, World! Python"
```

3. Lists: Lists are ordered collections of items. They can contain any type of data and are defined using square brackets. Lists support operations such as append, insert, remove, and indexing.

In [ ]:
```
l = [1, 2, 3, 4, 5]
print(l)
print(l[0]) # 1
print(l[1:3]) # [2, 3]
l.append(6)
print(l) # [1, 2, 3, 4, 5, 6]
l.insert(2,10)
print(l) # [1, 2, 10, 3, 4, 5, 6]
l.remove(10)
print(l) # [1, 2, 3, 4, 5, 6]
```

List comprehensions are a concise way to create lists in Python. They are a powerful tool for working with lists and other iterable data types, and are an alternative to using loops and the `append()` method to create lists.

Here is an example of using a for loop to create a list of squares of the numbers from 0 to 9:

In [ ]:
```
squares = []
for x in range(10):
    squares.append(x**2)
print(squares) # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

This can be simplified using a list comprehension as follows:

In [ ]:
```
squares = [x**2 for x in range(10)]
print(squares) # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List comprehensions can also include an optional `if` clause to filter the items included in the list. Here is an example of creating a list of only the even squares from the previous example:

In [ ]:
```
even_squares = [x**2 for x in range(10) if x % 2 == 0]
print(even_squares) # [0, 4, 16, 36, 64]
```

You can also use list comprehension with multiple for loops.

In [ ]:
```
matrix = [[1,2,3], [4,5,6], [7,8,9]]
flattened_list = [x for sublist in matrix for x in sublist]
print(flattened_list) # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List comprehensions can be a more efficient and readable way to create lists in Python, and are a good tool to have in your toolbox when working with data in Python.

4. Tuples: Python tuples are similar to lists, but are immutable, meaning that their contents cannot be modified after they are created. They are defined using parentheses.

In [2]:
```
# Example of tuple
t = (1, 2, 3, 4, 5)
print(t)
print(t[0]) # 1
print(t[1:3]) # (2, 3)
```

5. Dictionaries: Dictionaries are unordered collections of items, each consisting of a key-value pair. They are defined using curly braces {}. They are used to store key-value pairs, and can be indexed using keys.

In [ ]:
```
# Example of dictionary
d = { "one":1, "two":2, "three":3 }
print(d)
print(d["one"]) # 1
d["four"] = 4
print(d) # { "one":1, "two":2, "three":3, "four":4 }
del d["four"]
print(d) # { "one":1, "two":2, "three":3 }
```

6. Booleans: Python has a Boolean data type, which can be either True or False.

In [ ]:
```
# Example of booleans
flag = True
check = False
```

7. Sets: Sets are unordered collections of unique items. They are defined using curly braces {}. Sets do not allow duplicate elements, and support operations such as union, intersection, and difference.

In [ ]:
```
# Example of sets
s1 = {1, 2, 3}
s2 = {3, 4, 5}
print(s1)
print(s1.union(s2)) # {1, 2, 3, 4, 5}
print(s1.intersection(s2)) # {3}
print(s1.difference(s2)) # {1, 2}
```

It's also worth noting that Python is a dynamically typed language, which means that the data type of a variable can change at runtime, and that you don't need to specify the data type when you declare a variable.

Basic Mathematical Operations and Expressions

Python provides a wide range of mathematical operations and expressions that can be used to perform various types of calculations. Here are a few examples:

1. Arithmetic Operators: Python supports basic arithmetic operators such as addition (+), subtraction (-), multiplication (*), division (/), modulus (%), and floor division (//).

In [ ]:
```
# Example of arithmetic operators
a = 5
b = 2
```

```
# Addition
print(a + b)  # 7

# Subtraction
print(a - b)  # 3

# Multiplication
print(a * b)  # 10

# Division
print(a / b)  # 2.5

# Modulus
print(a % b)  # 1

# Floor Division
print(a // b)  # 2
```

1. Assignment Operators: Python supports several assignment operators that can be used to change the value of a variable.

In [ ]:
```
# Example of assignment operators
x = 5
x += 2 # x = x + 2
x -= 2 # x = x - 2
x *= 2 # x = x * 2
x /= 2 # x = x / 2
```

1. Relational Operators: Python supports relational operators that can be used to compare values, and that return a Boolean value.

In [ ]:
```
# Example of relational operators
x = 5
y = 2

print(x == y)  # False
print(x != y)  # True
print(x > y)  # True
print(x < y)  # False
print(x >= y)  # True
print(x <= y)  # False
```

1. Logical Operators: Python supports logical operators that can be used to combine relational expressions, and that return a Boolean value.

In [ ]:
```
# Example of logical operators
x = True
y = False

print(x and y)  # False
print(x or y)  # True
print(not x)  # False
```

1. Built-in functions Python has a wide range of built-in mathematical functions that can be used to perform various types of calculations. Some examples include:

In [ ]:
```python
# Example of built-in functions
import math

# square root
print(math.sqrt(16))  # 4

# power
print(math.pow(2,3))  # 8

# trigonometric functions
print(math.sin(math.radians(90)))  # 1.0

# logarithmic functions
print(math.log10(100))  # 2.0
```

These are just a few examples of the many mathematical operations and expressions that are available in Python. The best way to learn more is to practice using them in your own programs.

Exercises

Here are a few exercises to help you practice working with mathematical operations, data types, and expressions in Python:

1. Exercise 1: Write a program that prompts the user for two integers, and then performs the following operations using those integers: addition, subtraction, multiplication, division, modulus, and floor division. Print the results of each operation to the console.

2. Exercise 2: Write a program that prompts the user for a string, and then performs the following operations using that string: concatenation, repetition, and indexing. Print the results of each operation to the console.

3. Exercise 3: Write a program that prompts the user for a list of integers, and then performs the following operations using that list: appending, inserting, and removing elements. Print the results of each operation to the console.

4. Exercise 4: Write a program that prompts the user for a number, and then performs the following mathematical operations: square root, power, trigonometric and logarithmic functions. Print the results of each operation to the console.

These exercises should give you a good starting point for working with mathematical operations, data types, and expressions in Python. Remember that practice is key to becoming proficient in any programming language, so don't hesitate to experiment with different types of inputs and operations to see how they behave.

In [ ]:
```python

```