Python Comments

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

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

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

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

A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

```
#print("Hello, World!")
print("Cheers, Mate!")
```

Multiline Comments

Python does not have a syntax for multiline comments.

To add a multiline comment you could insert a # for each line:

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

1. Variable Names Must Start With a Letter or an Underscore (_):

- Valid: `name`, `_value`
- Invalid: `1name`, `@data`

---

2. Variable Names Can Contain Letters, Digits, and Underscores (_):

- Valid: `age_23`, `value_1`
- Invalid: `age#23`, `value-1`

---

3. Variable Names Are Case-Sensitive:

- `Name` and `name` are considered different variables.

---

4. Keywords Cannot Be Used as Variable Names:

Python's reserved words (e.g., `class`, `for`, `while`, `if`, `else`, etc.) cannot be used.

- Invalid: `if = 10`, `class = "Python"`

---

5. Variable Names Should Be Descriptive (Best Practice):

Use meaningful names for better readability and maintainability.

- Example: Use `user_name` instead of `x`.

---

6. No Spaces in Variable Names:

Use underscores (_) to separate words.

- Valid: `user_name`, `max_value`
- Invalid: `user name`, `max value`

---

7. Variables Can Be Reassigned:

Python variables are dynamic; you can assign a new value of any type to an existing variable.

```
x = 5      # integer
x = "five" # string
```

---

8. Follow PEP 8 Naming Guidelines (Optional but Recommended):

- Use lowercase words separated by underscores for variables (`snake_case`).
- Example: `total_cost`, `student_name`

• Get the Type

You can get the data type of a variable with the `type()` function.

```
x = 5

y = "John"

print(type(x))

print(type(y))
```

Single or Double Quotes?

String variables can be declared either by using single or double quotes:

```
x = "John"

# is the same as

x = 'John'
```

Python Variables - Assign Multiple Values

Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"

print(x)

print(y)

print(z)
```

Note: Make sure the number of variables matches the number of values, or else you will get an error.

One Value to Multiple Variables

And you can assign the *same* value to multiple variables in one line:

```
x = y = z = "Orange"

print(x)

print(y)

print(z)
```

Output Variables

The Python `print()` function is often used to output variables.

```
x = "Python is awesome"

print(x)
```

In the `print()` function, you output multiple variables, separated by a comma:

```
x = "Python"

y = "is"

z = "awesome"

print(x, y, z)
```

you can also use the `+` operator to output multiple variables:

```
x = "Python "

y = "is "

z = "awesome"

print(x + y + z)
```

Notice the space character after `"Python "` and `"is "`, without them the result would be "Pythonisawesome".

For numbers, the `+` character works as a mathematical operator:

```
x = 5

y = 10

print(x + y)
```

In the `print()` function, when you try to combine a string and a number with the `+` operator, Python will give you an error:

```
x = 5

y = "John"

print(x + y)
```

```
x = 5

y = "John"

print(x, y)
```

Python Numbers

Python Numbers

There are three numeric types in Python:

- `int`
- `float`
- `complex`

Variables of numeric types are created when you assign a value to them:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

#To verify the type of any object in Python, use the `type()` function:

```
print(type(x))
print(type(y))
print(type(z))
```

Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

Float can also be scientific numbers with an "e" to indicate the power of 10.

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

Complex

Complex numbers are written with a "j" as the imaginary part:

```
x = 3+5j

y = 5j

z = -5j

print(type(x))

print(type(y))

print(type(z))
```

Python - Modify Strings

Python has a set of built-in methods that you can use on strings.

Upper Case

The `upper()` method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())
```

Lower Case

The `lower()` method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())
```

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

The `strip()` method removes any whitespace from the beginning or the end:

```
a = "                                                Hello,World!"
print(a.strip())
```

Replace String

The `replace()` method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Python - String Concatenation

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Merge variable `a` with variable `b` into variable `c`:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

Example

To add a space between them, add a `" "`:

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

But we can combine strings and numbers by using *f-strings* or the `format()` method!

F-Strings

F-String was introduced in Python 3.6, and is now the preferred way of formatting strings.

To specify a string as an f-string, simply put an `f` in front of the string literal, and add curly brackets `{}` as placeholders for variables and other operations.

```
age = 36
txt = f"My name is John, I am {age}"
print(txt)
```

Placeholders and Modifiers

A placeholder can contain variables, operations, functions, and modifiers to format the value.

```
price = 59
txt = f"The price is {price} dollars"
print(txt)
```

A placeholder can include a *modifier* to format the value.

A modifier is included by adding a colon`:` followed by a legal formatting type, like `.2f` which means fixed point number with 2 decimals:

```
price = 59

txt = f"The price is {price:.2f} dollars"

print(txt)
```

A placeholder can contain Python code, like math operations:

Perform a math operation in the placeholder, and return the result:

```
txt = f"The price is {20 * 59} dollars"
print(txt)
```

New Line -

txt = "Hello\nWorld!"
print(txt)

- **Example for each escape sequence in Python**

1. Single Quote (\'):

Allows a single quote inside a string enclosed by single quotes.

```
print('It\'s a sunny day!')
# Output: It's a sunny day!
```

---

2. Backslash (\\):

Prints a backslash character.

```
print("This is a backslash: \\")
# Output: This is a backslash: \
```

---

3. New Line (\n):

Moves the text after it to a new line.

```
print("Hello\nWorld")
# Output:
# Hello
# World
```

4. Carriage Return (`\r`):

Moves the cursor to the beginning of the line, overwriting the text.

```
print("Hello\rWorld")
# Output: World
```

---

5. Tab (`\t`):

Adds a horizontal tab space.

```
print("Hello\tWorld")
# Output: Hello    World
```

---

6. Backspace (`\b`):

Removes the previous character.

```
print("Helloo\b World")
# Output: Hello World
```

- **float**

```
#Define a float variable
price = 19.99

# Print the value
print("The price is:", price)

# Perform operations with floats
discount = 5.50
final_price = price - discount
print("Final price after discount is:", final_price)
```

Here are 10 additional examples to help you understand how floats work in Python:

**1. Declaring and Printing a Float**
```python
temperature = 36.6
print("Body temperature is:", temperature)
 # Output: Body temperature is: 36.6
```

**2. Float Arithmetic**
```python
a = 5.5
b = 2.2
result = a + b
print("Sum is:", result)
 # Output: Sum is: 7.7
```

**3. Multiplication with Floats**
```python
length = 4.5
width = 3.2
area = length * width
print("Area of rectangle is:", area)
 # Output: Area of rectangle is: 14.4
```

**4. Division of Floats**
```python
numerator = 10.0
denominator = 4.0
result = numerator / denominator
print("Division result is:", result)
 # Output: Division result is: 2.5
```

---

5. Using Floats in Power Calculations
```python
base = 2.5
exponent = 3
result = base ** exponent
print("Power result is:", result)
# Output: Power result is: 15.625
```

---

6. Combining Integers and Floats
```python
int_value = 10
float_value = 2.5
result = int_value * float_value
print("Result of multiplying integer and float is:", result)
# Output: Result of multiplying integer and float is: 25.0
```

---

7. Rounding a Float
```python
value = 3.14159
rounded_value = round(value, 2)
print("Rounded value is:", rounded_value)
# Output: Rounded value is: 3.14
```

---

8. Converting Float to Integer

```
float_value = 7.9
int_value = int(float_value)
print("Converted to integer:", int_value)
# Output: Converted to integer: 7
```

9. Negative Floats

```
balance = -150.75
print("Account balance is:", balance)
# Output: Account balance is: -150.75
```

10. Floats in a List
```
prices = [19.99, 25.5, 100.75, 3.5]
total = sum(prices)
print("Total of all prices:", total)
# Output: Total of all prices: 149.74
```