Day 1:

Python was created in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language. It was originally designed as a scripting language for the Amoeba operating system, but it has since become a popular general-purpose language.

Python is known for its simplicity, readability, and its concise syntax. It is also a very versatile language, which makes it a good choice for a wide range of applications.

Here are some of the key events in the history of Python programming:

- 1989: Guido van Rossum starts working on Python at CWI.
- 1991: Python 0.9.0 is released.
- 1994: Python 1.0 is released.
- 2000: Python 2.0 is released.
- 2008: Python 3.0 is released.
- 2020: Python 2.7 is discontinued.

Python 3 is a major revision of the language that is not completely backward-compatible with Python 2. This means that code written for Python 2 will not run without modification in Python 3. However, there are many tools available to help you convert your code from Python 2 to Python 3.

Python is a general-purpose, high-level programming language. It is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented and functional programming.

Python is a popular language for a variety of tasks, including:

- Web development
- Data science
- Machine learning
- Artificial intelligence
- System administration
- Scientific computing
- Game development

Here are some other organizations that use Python:

- The European Space Agency (ESA)
- The United States Department of Defense (DoD)
- The National Institutes of Health (NIH)
- The World Health Organization (WHO)
- The New York Stock Exchange (NYSE)
- The Bank of England
- The International Monetary Fund (IMF)

Python is known for its simplicity, readability, and its concise syntax. It is also a very versatile language, which makes it a good choice for a wide range of applications.

Here are some of the benefits of using Python:

- Easy to learn: Python has a simple syntax that is easy for beginners to learn.
- Versatile: Python can be used for a wide range of tasks, from web development to data science.
- Powerful: Python is a powerful language that can be used to create complex applications.
- Open source: Python is an open source language, which means that it is free to use and modify.
- Community: There is a large and active community of Python developers who can help you learn and use the language.

Python is a popular language used by many companies and organizations around the world. Here are some of the most well-known companies that use Python:

- Google
- Netflix
- Spotify
- NASA
- Tesla
- Facebook
- Amazon
- Dropbox
- Instagram

- Reddit
- Uber

If you are looking for a programming language that is easy to learn, versatile, and powerful, then Python is a good choice.

Here is an example of a Python program that prints the message "Hello, world!":
Python
print("Hello, world!")

Variables in Python

A variable is a named memory location that can be used to store data. In Python, variables are created when they are first assigned a value.
Python
# Create a variable named `my_name` and assign it the value `John Doe`
my_name = "John Doe"

# Print the value of the variable
print(my_name)

The output of the above code is:
John Doe

Variable names in Python can be any length and can consist of lowercase and uppercase letters, digits, and the underscore character (_). An additional restriction is that, although a variable name can contain digits, the first character of a variable name cannot be a digit.
Python
# Valid variable names
my_name = "John Doe"

# Invalid variable names
1name = "John Doe"
name1 = "John Doe"

Python has different types of variables, each of which can store different types of data. The most common types of variables in Python are:

- Numeric variables: These variables can store numbers.
- String variables: These variables can store text.
- List variables: These variables can store a collection of values.
- Tuple variables: These variables can store a collection of values, but the values cannot be changed.
- Dictionary variables: These variables can store a collection of key-value pairs.
- Set variables: These variables can store a collection of unique values.

Example Output

When you run your program, it should print the following:

```
Day 1 - String Manipulation
String Concatenation is done with the "+" sign.
e.g. print("Hello " + "world")
New lines can be created with a backslash and n.
```

The type of a variable is determined by the value that is assigned to it. For example, the following code creates a numeric variable and a string variable:
Python
# Create a numeric variable
my_number = 10

# Create a string variable
my_name = "John Doe"

The variable my_number is a numeric variable because it is assigned the value 10, which is a number. The variable my_name is a string variable because it is assigned the value "John Doe", which is a string.

Variables can be used to store data temporarily or permanently. Temporary variables are created within a function or a block of code and are deleted when the function or block of code is finished executing. Permanent variables are created outside of any function or block of code and remain in memory until the program is terminated.

Variables are an important part of programming in Python. They allow you to store data and reuse it throughout your program. By understanding how variables work, you can write more efficient and reusable code.

Here are some additional things to keep in mind about variables in Python:

- The value of a variable can be changed at any time.
- Multiple variables can have the same name, but they must be of different types.
- Variables are case-sensitive.
- Variables cannot be reused within the same scope.

Rules for Naming an Identifier

- Identifiers cannot be a keyword.
- Identifiers are case-sensitive.
- It can have a sequence of letters and digits. However, it must begin with a letter or `_`. The first letter of an identifier cannot be a digit.
- It's a convention to start an identifier with a letter rather `_`.
- Whitespaces are not allowed.
- We cannot use special symbols like **!**, **@**, **#**, **$**, and so on.

Reserved keywords are special words that have a predefined meaning in Python. They cannot be used as variable names, function names, or any other identifier.

Here are the 33 reserved keywords in Python:

Python Keywords List

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

Some Valid and Invalid Identifiers in Python

Valid Identifiers	Invalid Identifiers
score	@core
return_value	return
highest_score	highest score
name1	1name
convert_to_string	convert to_string

Things to Remember

Python is a case-sensitive language. This means, `Variable` and `variable` are not the same.

Always give the identifiers a name that makes sense. While `c = 10` is a valid name, writing `count = 10` would make more sense, and it would be easier to figure out what it represents when you look at your code after a long gap.

Multiple words can be separated using an underscore, like `this_is_a_long_variable`.

Python Data Types

In computer programming, data types specify the type of data that can be stored inside a variable. For example,

```
num = 24
```

Here, **24** (an integer) is assigned to the `num` variable. So the data type of `num` is of the `int` class.

Python Data Types

Data Types	Classes	Description
Numeric	int, float, complex	holds numeric values
String	str	holds sequence of characters
Sequence	list, tuple, range	holds collection of items
Mapping	dict	holds data in key-value pair form
Boolean	bool	holds either `True` or `False`
Set	set, frozeenset	hold collection of unique items

Python Numeric Data type

In Python, numeric data type is used to hold numeric values.

Integers, floating-point numbers and complex numbers fall under Python numbers category. They are defined as `int`, `float` and `complex` classes in Python.

- `int` - holds signed integers of non-limited length.
- `float` - holds floating decimal points and it's accurate up to **15** decimal places.
- `complex` - holds complex numbers.

We can use the `type()` function to know which class a variable or a value belongs to.

Let's see an example,

```
num1 = 5
print(num1, 'is of type', type(num1))

num2 = 2.0
print(num2, 'is of type', type(num2))

num3 = 1+2j
print(num3, 'is of type', type(num3))
```

Run Code »

Output

```
5 is of type <class 'int'>
2.0 is of type <class 'float'>
(1+2j) is of type <class 'complex'>
```

String Manipulation

---

string1 = "This is a string."
string2 = 'This is also a string.'
string3 = "1234567890"
string4 = "Hello, world!"

What is a string in Python?

Why is string manipulation important in Python?
What are some of the common problems that programmers encounter when manipulating strings?
How can string manipulation be used to solve real-world problems?
I can also answer any questions that you have about string manipulation in Python.

Python

```
string1 = "This is a string."
string2 = 'This is also a string.'
string3 = "1234567890"
string4 = "Hello, world!"
```

Use code with caution. Learn more

The different string operations that can be performed in Python.
There are many string operations that can be performed in Python. Here are a few examples:

Concatenating strings: This is the process of joining two or more strings together.
Here is another example of string concatenation:

Python

```
str1 = "This is a string."
str2 = " And this is another string."

print(str1 + " " + str2)
```

Use code with caution. Learn more

Splitting strings: This is the process of breaking a string into smaller strings.

```
sentence = "This is a sample sentence."
words = sentence.split()
print("Split using default whitespace delimiter:")
print(words)  # Output: ['This', 'is', 'a', 'sample', 'sentence.']
```

```
csv_data = "John,Doe,30,New York"
fields = csv_data.split(',')
print("\nSplit using a custom delimiter (','):")
print(fields)  # Output: ['John', 'Doe', '30', 'New York']
```

```
# Using str.split() with maxsplit parameter
data = "apple,banana,cherry,dates,fig"
fruits = data.split(',', 2)
print("\nSplit using maxsplit (max of 2 splits):")
print(fruits)  # Output: ['apple', 'banana', 'cherry,dates,fig']
```

```
sentence = "This is a sample sentence. It contains words."
tokens = sentence.split(' ')
print("\nTokenizing a sentence:")
print(tokens)  # Output: ['This', 'is', 'a', 'sample', 'sentence.', 'It', '
```

Python

```
str1 = "This_is_a_string_with_underscores."
delimiter = "_"

print(split_string(str1, delimiter))
```

Use code with caution. Learn more

```
['This', 'is', 'a', 'string', 'with', 'underscores.']
```

Searching strings: This is the process of finding a specific substring within a string.

Python

```
str1 = "This is a string with a substring."
substring = "substring"

index = str1.find(substring)
```

```
The substring 'substring' is found at index 24.
```

Replacing strings: This is the process of replacing a substring within a string with another substring.

Python

```
str1 = "This is a string with a substring."
substring = "substring"
new_substring = "new substring"

new_str = str1.replace(substring, new_substring)

print(new_str)
```

Use code with caution. Learn more

```
This is a string with a new substring.
```

python Copy code

```
original_text = "I love apples. Apples are delicious."
new_text = original_text.replace("apples", "bananas")
print(new_text)
```

In this example, we use the `replace()` method to replace all occurrences of "apples" with "bananas" in the `original_text`. When you run this code, it will produce the following output:

css Copy code

```
I love bananas. Bananas are delicious.
```

Formatting strings: This is the process of changing the appearance of a string.

Python

```
str1 = "The value is {0}."

value = 10

new_str = str1.format(value)

print(new_str)
```

Use code with caution. Learn more

```
The value is 10.
```

```
# Using positional arguments with .format()
name = "Charlie"
age = 35
formatted_string = "My name is {0} and I am {1} years old.".format(name, age)
print(formatted_string)
```

The built-in string methods in Python.

Python has a number of built-in string methods that can be used to manipulate strings. Here are a few examples:

str.upper(): This method converts a string to uppercase.

Python

```
str1 = "This is a string."

upper_str = str1.upper()

print(upper_str)
```

Use code with caution. Learn more

```
THIS IS A STRING.
```

str.lower(): This method converts a string to lowercase.

Python

```
str1 = "This is a string."

lower_str = str1.lower()

print(lower_str)
```

Use code with caution. Learn more

```
this is a string.
```

str.strip(): This method removes whitespace from the beginning and end of a string.

Python

```
str1 = " This is a string. "

stripped_str = str1.strip()

print(stripped_str)
```

Use code with caution. Learn more

```
This is a string.
```

str.replace(): This method replaces a substring within a string with another substring.

```python
str1 = "This is a string."

new_str = str1.replace("string", "word")

print(new_str)
```

Use code with caution. Learn more

```
This is a word.
```

str.format(): This method formats a string according to a specific format string.

```python
str1 = "The value is {0}."

value = 10

new_str = str1.format(value)

print(new_str)
```

Use code with caution. Learn more

```
The value is 10.
```

Input and Print Functions

The input() function takes a string as input and returns it as a string. For example, the following code will ask the user to enter their name and then print it back to them:

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

Use code with caution. Learn more

```python
age = input("Enter your age: ")
print("You are " + age + " years old.")
```

Use code with caution. Learn more

Sure, I can help you with that.

In Python, the input() function always returns a string, even if the user enters a number. If you want to convert the input to a different data type, you can use the following built-in functions:
0-9
int(): Converts a string to an integer.
0.xxxx
float(): Converts a string to a floating-point number.
a-z

“2” +”3”=23

str(): Converts a number to a string.
bool(): Converts a string to a Boolean value.

Python

```
number = input("Enter a number: ")
integer_number = int(number)
```

Use code with caution. Learn more

- Convert a string to a floating-point number:

Python

```
number = input("Enter a number: ")
floating_number = float(number)
```

Use code with caution. Learn more

- Convert a number to a string:

Python

```
number = 123
string_number = str(number)
```

Use code with caution. Learn more

- Convert a string to a Boolean value:

Python

```
answer = input("Yes or no? ")
boolean_answer = bool(answer)
```

Use code with caution. Learn more