Variables

A variable is a data container.

It can contain 1 information per time.

age = 30

Variables

- Start with letter or _underscore
- Cannot start with a number
- Unique word in the code
- Only Alpha-numeric chars

  (a-z or A-Z or 0-9)
- No Python keywords

Variables

best_practices

Assign a value to use **a variable**.

my_first_name = "Lisa"
my_last_name = "Hennecart"

```
my_age = 27

print("My name is", my_first_name)
 # My name is Lisa
print("My name is", my_first_name,
m_last_namey)
# My name is Lisa Hennecart
```

```
my_name = "Lisa"
my_age = 27

print("My name is", my_name)

print("My age is", my_age)
# My age is 27
```

Assigning values to Variables in Python

As we can see from the above example, we use the assignment operator = to assign a value to a variable.

```
# assign value to site_name variable
course = 'python'

print(course)

# Output: python
```

Changing the Value of a Variable in Python

```
course = ‘python'
print(course)

# assigning a new value to site_name
course = ‘learn_python'

print(course)

#Output: learn_python
```

Example: Assigning multiple values to multiple variables

```
a, b, c = 5, 3.2, 'Hello'

print (a)  # prints 5
print (b)  # prints 3.2
print (c)  # prints Hello
```

Please Do Exercise 1

Data Types in Python

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, frozenset	hold collection of unique items

Python Numbers

The number data type are used to store the numeric values.
Python supports integers, floating-point numbers and complex numbers. 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 = 5.42
print(num2, 'is of type', type(num2))

num3 = 8+2j
print(num3, 'is of type', type(num3))
```

```
5 is of type <class 'int'>
5.42 is of type <class 'float'>
(8+2j) is of type <class 'complex'>
```

Python Strings

In Python, a string is a sequence of characters.
For example, "hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', and 'o'.

We use single quotes or double quotes to represent a string in Python.
For example

```
# create a string using double quotes
string1 = "Python programming"
# create a string using single quotes string1 = 'Python
programming'
```

Here, we have created a string variable named **string1**. The variable is initialized with the string "Python Programming".

```
# create string type variables
name = "Python"
print(name)
message = "I love Python."
print(message)
```

```
Python
I love Python.
```

Python List

There are many built-in types in Python that allow us to group and store multiple items. Python lists are the most versatile among them.
For example, we can use a Python list to store a playlist of songs so that we can easily add, remove, and update songs as needed.

Create a Python List

We create a list by placing elements inside square brackets [], separated by commas.
For example,

```
 # a list of three elements
ages = [19, 26, 29]
print(ages)

# Output: [19, 26, 29]
```

Python Tuple

A tuple is a collection similar to a Python list. The primary difference is that we cannot modify a tuple once it is created.

Create a Python Tuple

We create a tuple by placing items inside parentheses ().
For example,

```
numbers = (1, 2, -5)
print(numbers)

# Output: (1, 2, -5)
```

Python Dictionary

A Python dictionary is a collection of items, similar to lists and tuples. However, unlike lists and tuples, each item in a dictionary is a **key-value** pair (consisting of a key and a value).

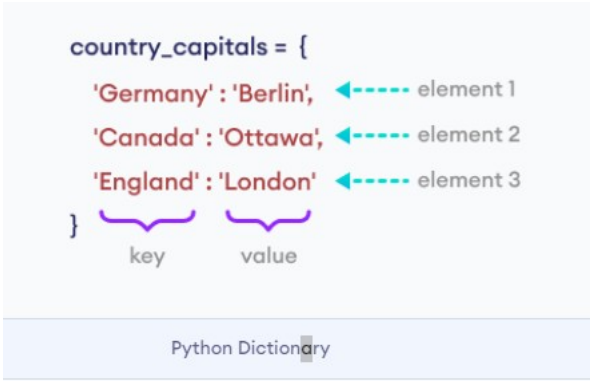
eate a Dictionary

e create a dictionary by placing key: value pairs inside curly brackets {}, separated by commas.
r example,

```
# creating a dictionary
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa",
  "England": "London"
}

# printing the dictionary
print(country_capitals)
```

```
{'Germany': 'Berlin', 'Canada': 'Ottawa', 'England': 'London'}
```



Python Dictionary

Python Type Conversion

In programming, type conversion is the process of converting data of one type to another.

For example: converting int data to str.

There are two types of type conversion in Python.

- Implicit Conversion - automatic type conversion
- Explicit Conversion - manual type conversion

Python Implicit Type Conversion

In certain situations, Python automatically converts one data type to another. This is known as implicit type conversion.

Example 1: Converting integer to float

Let's see an example where Python promotes the conversion of the lower data type (integer) to the higher data type (float) to avoid data loss.

```
integer_number = 123
float_number = 1.23

new_number = integer_number + float_number

# display new value and resulting data type
print("Value:",new_number)
print("Data Type:",type(new_number))
```

In the above example, we have created two variables:
integer_number and *float_number* of *int* and *float*
type respectively.
Then we added these two variables and stored the result in new_number.
It is because Python always converts smaller data types to larger data types to avoid the

Note:

- We get TypeError, if we try to add str and int. For example, '12' + 23. Python is not able to use Implicit Conversion in such conditions.
- Python has a solution for these types of situations which is known as Explicit Conversion.

Explicit Type Conversion

In Explicit Type Conversion, users convert the data type of an object to required data type.

We use the built-in functions like int(), float(), str(), etc to perform explicit type conversion.

This type of conversion is also called typecasting because the user casts (changes) the data type of the objects.

Example 2: Addition of string and integer Using Explicit Conversion

```
num_string = '12'
num_integer = 23

print("Data type of num_string before Type
Casting:",type(num_string))

# explicit type conversion
num_string = int(num_string)

print("Data type of num_string after Type
Casting:",type(num_string))

num_sum = num_integer + num_string

print("Sum:",num_sum)
print("Data type of num_sum:",type(num_sum))
```

```
Data type of num_string before Type Casting: <class 'str'>
Data type of num_string after Type Casting: <class 'int'>
Sum: 35 Data type of num_sum: <class 'int'>
```

Key Points to Remember

1. Type Conversion is the conversion of an object from one data type to another data type.
2. Implicit Type Conversion is automatically performed by the Python interpreter.
3. Python avoids the loss of data in Implicit Type Conversion.
4. Explicit Type Conversion is also called Type Casting, the data types of objects are converted using predefined functions by the user.
5. In Type Casting, loss of data may occur as we enforce the object to a specific data type.

Python Functions

A function is a block of code that performs a specific task.

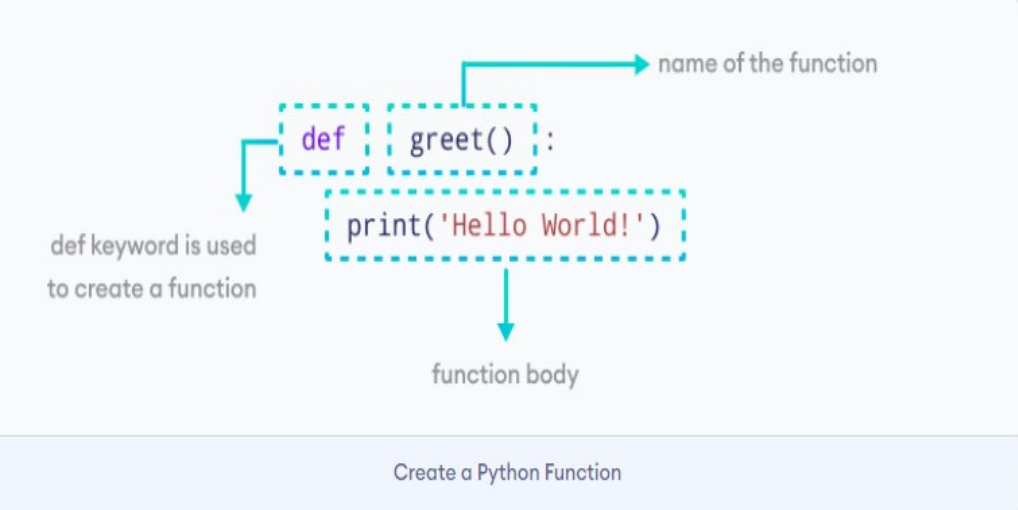
Suppose we need to create a program to make a circle and color it. We can create two functions to solve this problem:

function to create a circle
function to color the shape

Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

Let's create our first function.

```
def greet():
    print('Hello World!')
```



Calling a Function

In the above example, we have declared a function named greet().

```
def greet():
    print('Hello World!')
```

Function Call

```
greet()
```

Example: Python Function Call

```
def greet():
    print('Hello World!')

# call the function
greet()

print('Outside function')
```

```
Hello World!
Outside function
```

In the above example, we have created a function named `greet()`. Here's how the control of the program flows:

Working of Python Function

Arithmetic operations

```
# Arithmetic Operations
x = 10
y = 5
addition = x + y  # Addition
subtraction = x - y  # Subtraction
multiplication = x * y  # Multiplication
division = x / y  # Division
integer_division = x // y  # Integer
Division (floor division)
modulo = x % y  # Modulo
(remainder)
exponentiation = x ** y  #
```

```
print("Arithmetic Operations:")
print("Addition:", addition)
print("Subtraction:", subtraction)
print("Multiplication:", multiplication)
print("Division:", division)
print("Integer Division:",
integer_division)
print("Modulo:", modulo)
print("Exponentiation:",
exponentiation)
```

```
# Increment and Decrement
a = 5
a += 1  # Increment
b = 10
b -= 1  # Decrement

print("\nIncrement and Decrement:")
print("Increment a:", a)
print("Decrement b:", b)

# Swap
a = 5
b = 10
a, b = b, a  # Swap values of a and b

print("\nSwap a and b:")
print("a after swap:", a)
print("b after swap:", b)
```

String operations

```
# String Operations
string1 = "Hello"
string2 = "World“
string3 = "Python "
concatenation = string1 + " " + string2  # String Concatenation
repetition = string3 * 3  # String Repetition

print("\nString Operations:")
print("String Concatenation:", concatenation)
print("String Repetition:", repetition)
```

```
String Operations:
String Concatenation: Hello World
String Repetition: Python Python Python
```