Introduction to Python Programming

Module1

What is Python?

- Python is a widely-used, interpreted, object-oriented, and high-level programming language with dynamic semantics, used for general-purpose programming.
- **Python is a high-level programming language which is:**
- **Interpreted:** Python is processed at runtime by the interpreter.
- **Interactive:** You can use a Python prompt and interact with the interpreter directly to write your programs.
- **Object-Oriented:** Python supports Object-Oriented technique of programming.
- **Beginner’s Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications.

History of Python

- Python was conceptualized by **Guido Van Rossum** in the late **1980s**.
- Rossum published the first version of Python code (0.9.0) in February **1991** at the CWI (Centrum Wiskunde& Informatica) in the Netherlands , Amsterdam.
- Rossum chose the name "**Python**", since he was a big fan of Monty Python's Flying Circus.
- Guido van Rossum is a Dutch programmer best known as the author of the Python programming language
- Python is now maintained by a core development team at the institute, although Rossum still holds a vital role in directing its progress.

What makes Python special?

  - How does it happen that programmers, young and old, experienced and novice, want to use it? How did it happen that large companies adopted Python and implemented their flagship products using it?
  - There are many reasons – we've listed some of them already, but let's enumerate them again in a more practical manner:
  - it's **easy to learn** – the time needed to learn Python is shorter than for many other languages; this means that it's possible to start the actual programming faster;
  - it's **easy to teach** – the teaching workload is smaller than that needed by other languages; this means that the teacher can put more emphasis on general (language-independent) programming techniques, not wasting energy on exotic tricks, strange exceptions and incomprehensible rules;
  - it's **easy to use** for writing new software – it's often possible to write code faster when using Python;
  - it's **easy to understand** - it's also often easier to understand someone else's code faster if it is written in Python;
  - it's **easy to obtain, install and deploy** – Python is free, open and multiplatform; not all languages can boast that.
- **Of course, Python has its drawbacks, too:**
  - it's not a speed demon – Python does not deliver exceptional performance;
  - in some cases it may be resistant to some simpler testing techniques – this may mean that debugging Python code can be more difficult than with other languages; fortunately, making mistakes is also harder in Python.

Features of Python

1. Easy to Code
Python is a very high-level programming language, yet it is effortless to learn. Learning the basic Python syntax is very easy, as compared to other popular languages like C, C++, and Java.
2. Easy to Read
Python code looks like simple English words. There is no use of semicolons or brackets, and the indentations define the code block. You can tell what the code is supposed to do simply by looking at it.
3. Free and Open-Source
Python is developed under an OSI-approved open source license. Hence, it is completely free to use, even for commercial purposes.
4. Dynamically Typed
Many programming languages need to declare the type of the variable before runtime. With Python, the type of the variable can be decided during runtime. This makes Python a dynamically typed language.
For example, if you have to assign an integer value 20 to a variable “x”, you don’t need to write int x = 20. You just have to write x = 20.

4. Robust Standard Library

- Python has an extensive standard library available for anyone to use. This means that programmers don’t have to write their code for every single thing unlike other programming languages.

  5. Interpreted

- When a programming language is interpreted, it means that the source code is executed line by line, and not all at once. Programming languages such as C++ or Java are not interpreted, and hence need to be compiled first to run them.

  6. Portable

- Python is portable in the sense that the same code can be used on different machines. As such, there is no need to write a program multiple times for several platforms.

  7. Object-Oriented and Procedure-Oriented

- A programming language is object-oriented if it focuses design around data and objects, rather than functions and logic. On the contrary, a programming language is procedure-oriented if it focuses more on functions (code that can be reused). One of the critical Python features is that it supports both object-oriented and procedure-oriented programming.

8. Extensible

- A programming language is said to be extensible if it can be extended to other languages. Python code can also be written in other languages like C++, making it a highly extensible language.
  9. Expressive
- Python needs to use only a few lines of code to perform complex tasks. For example, to display Hello World, you simply need to type one line - print(“Hello World”).
  10. Support for GUI
- One of the key aspects of any programming language is support for GUI or Graphical User Interface. A user can easily interact with the software using a GUI. Python offers various toolkits, such as Tkinter, wxPython and JPython, which allows for GUI's easy and fast development.
  12. High-level Language
- Python is a high-level programming language because programmers don’t need to remember the system architecture, nor do they have to manage the memory.
  13. Simplify Complex Software Development
- Python can be used to develop both desktop and web apps and complex scientific and numerical applications. Python's data analysis features help you create custom big data solutions without so much time and effort. You can also use the Python data visualization libraries and APIs to present data in a more appealing way.

Applications of Python

1. Embedded scripting language: Python is used as an embedded scripting language for various testing/ building/ deployment/ monitoring frameworks, scientific apps, and quick scripts.
2. 3D Software:3D software like Maya uses Python for automating small user tasks, or for doing more complex integration such as talking to databases and asset management systems.
3. Web development: Python is an easily extensible language that provides good integration with database and other web standards.
4. GUI-based desktop applications: Simple syntax, modular architecture, rich text processing tools and the ability to work on multiple operating systems makes Python a preferred choice for developing desktop-based applications.
5. Image processing and graphic design applications: Python is used to make 2D imaging software such as Inkscape, GIMP, Paint Shop Pro and Scribus.

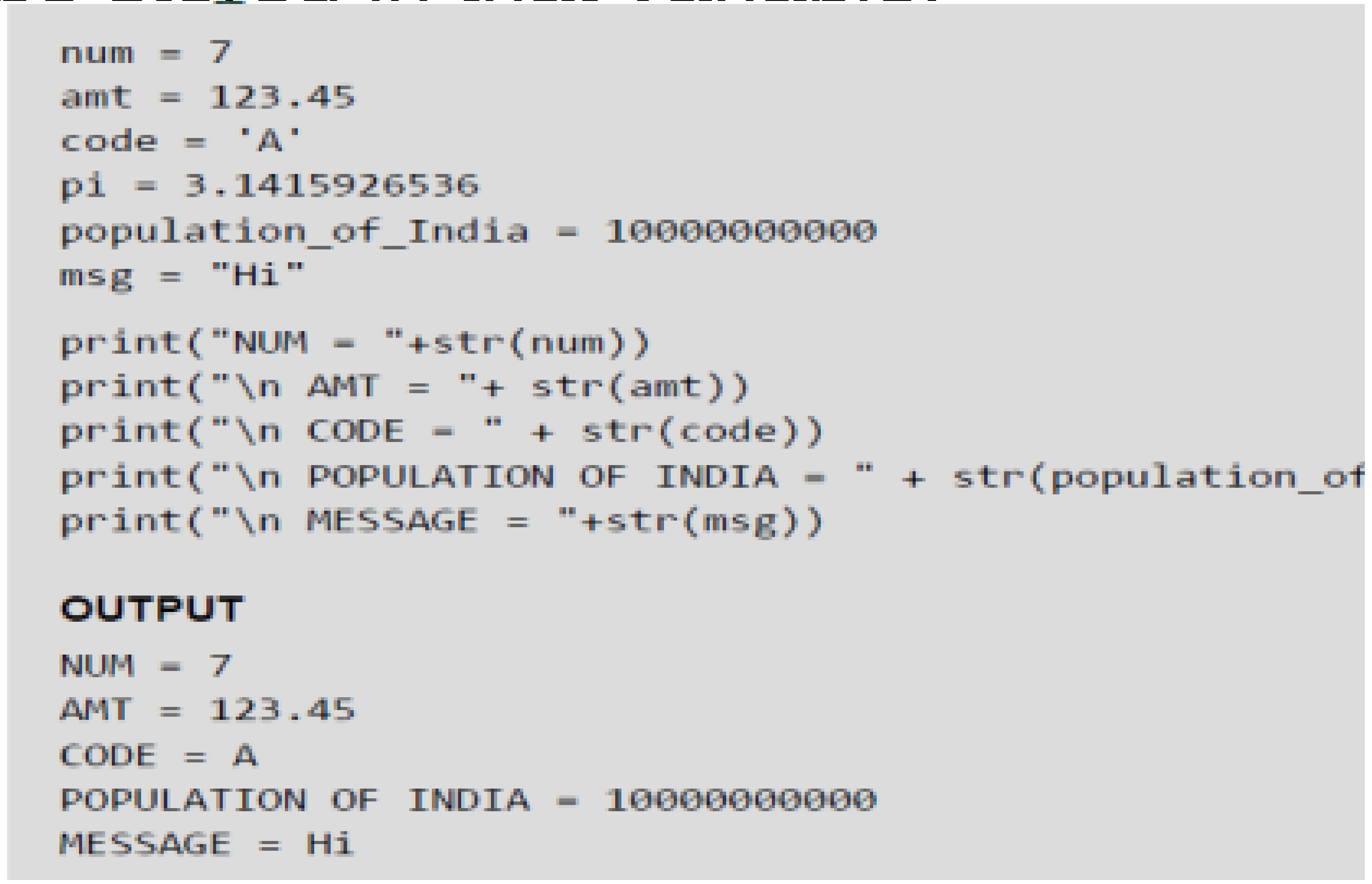
6. Scientific and computational applications: Features like high speed, productivity and availability of tools, such as Scientific Python and Numeric Python, have made Python a preferred language to perform computation and processing of scientific data. 3D modeling software, such as FreeCAD, and finite element method software, like Abaqus, are coded in Python.
7. Games: Python has various modules, libraries, and platforms that support development of games. Games like Civilization-IV, Disney's Toontown Online, Vega Strike, etc. are coded using Python.
8. Enterprise and business applications: Simple and reliable syntax, modules and libraries, extensibility, scalability together make Python a suitable coding language for customizing larger applications. For example, Reddit which was originally written in Common Lips, was rewritten in Python in 2005. A large part of Youtube code is also written in Python

Variables and Identifiers

- Variable means its value can vary. You can store any piece of information in a variable. Variables are nothing but just parts of your computer’s memory where information is stored. To be identified easily, each variable is given an appropriate name.
- Identifiers are names given to identify something. This something can be a variable, function, class, module or other object.
- For naming any identifier, there are some basic rules like:
  • The first character of an identifier must be an underscore ('_') or a letter (upper or lowercase).
  • The rest of the identifier name can be underscores ('_'), letters (upper or lowercase), or digits (0-9).
  • Identifier names are case-sensitive. For example, myvar and myVar are not same.

Assigning or Initializing Values to Variables

- In Python, programmers need not explicitly declare variables to reserve memory space. The declaration is done automatically when a value is assigned to the variable using the equal sign (=). The operand on the left side of equal sign is the name of the variable and the operand on its right side is the value to be stored in that variable.



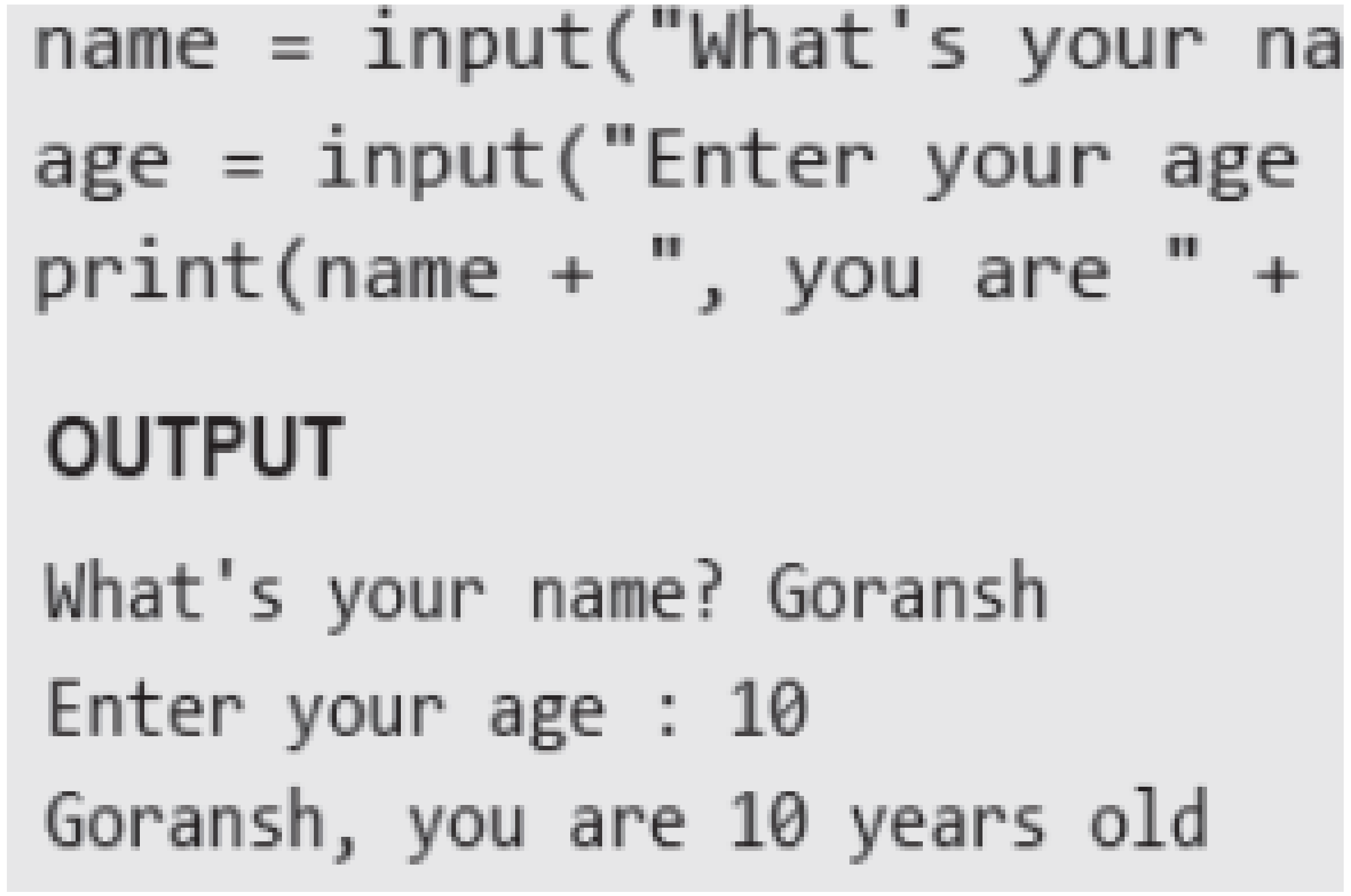
```
num = 7
amt = 123.45
code = 'A'
pi = 3.1415926536
population_of_India = 10000000000
msg = "Hi"
print("NUM = "+str(num))
print("\n AMT = "+ str(amt))
print("\n CODE = " + str(code))
print("\n POPULATION OF INDIA = " + str(population_of
print("\n MESSAGE = "+str(msg))
```

OUTPUT

```
NUM = 7
AMT = 123.45
CODE = A
POPULATION OF INDIA = 10000000000
MESSAGE = Hi
```

Accepting Input from Console: Input()

- User enters the values in the Console and that value is then used in the program as it is required. To take input from the user we make use of a built-in function *input()*.
-



```
name = input("What's your na
age = input("Enter your age
print(name + ", you are " +
```

OUTPUT

```
What's your name? Goransh
Enter your age : 10
Goransh, you are 10 years old
```

- We can also typecast this input to integer, float, or string by specifying the input() function inside the type.
- **1. Typecasting the input to Integer:** There might be conditions when you might require integer input from the user/Console, the following code takes two input(integer/float) from the console and typecasts them to an integer then prints the sum.
- **Example:**

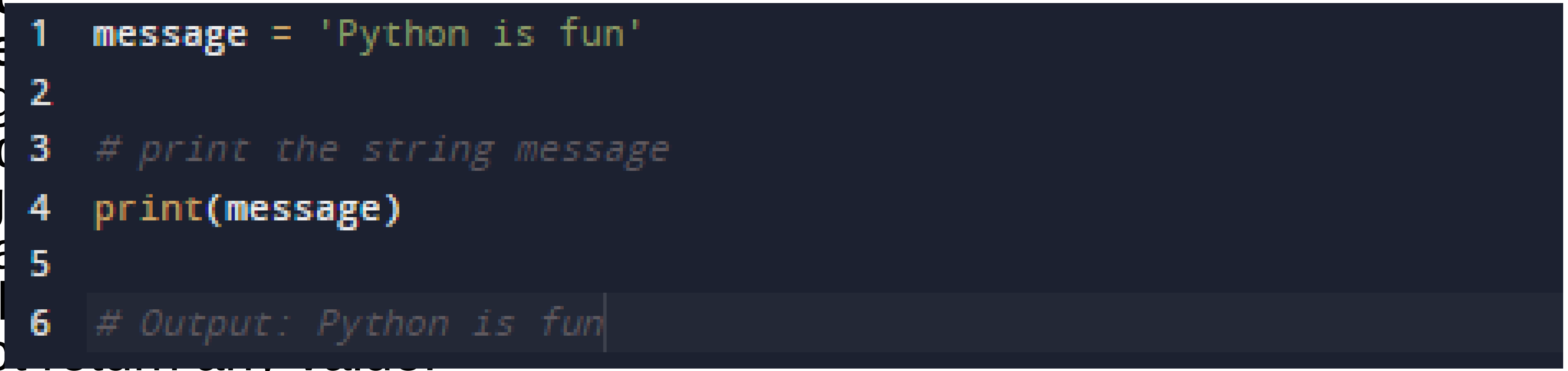
```
num1 = int(input())
num2 = int(input())
print(num1 + num2)
```

- **Typecasting the input to Float:** To convert the input to float the following code will work out.
- **Example:**

```
num1 = float(input())
num2 = float(input())
print(num1 + num2)
```

Output using print() function

- The print() function prints the given object to the standard output device (screen) or to the text stream file.
- **Example:**
- **What doe**
- Takes arg
  resulting c
- **What arg**
- Print() is a
- **What val**
- It does no



```python
message = 'Python is fun'

# print the string message
print(message)

# Output: Python is fun
```

String Literals

- String literals in python’s print statement are primarily used to format or design how a specific string appears when printed using the print() function.
- **\n :** This string literal is used to add a new blank line while printing a statement.
- **“” :** An empty quote (“”) is used to print an empty line.
- **Example:**
  ```
  print("GeeksforGeeks \n is best for DSA Content.")
  ```
- **Output:**
  ```
  GeeksforGeeks
   is best for DSA Content.
  ```

Print end command

- By default, print function in Python ends with a newline. This function comes with a parameter called ‘end.’ The default value of this parameter is ‘\n,’ i.e., the new line character. You can end a print statement with any character or string using this parameter. This is available in only in Python 3+
- Example:
- print ("Welcome to", end = ' ')
- print (“KJSCE", end = '!')
- **Output:**
- Welcome to KJSCE!

Print Option : sep=

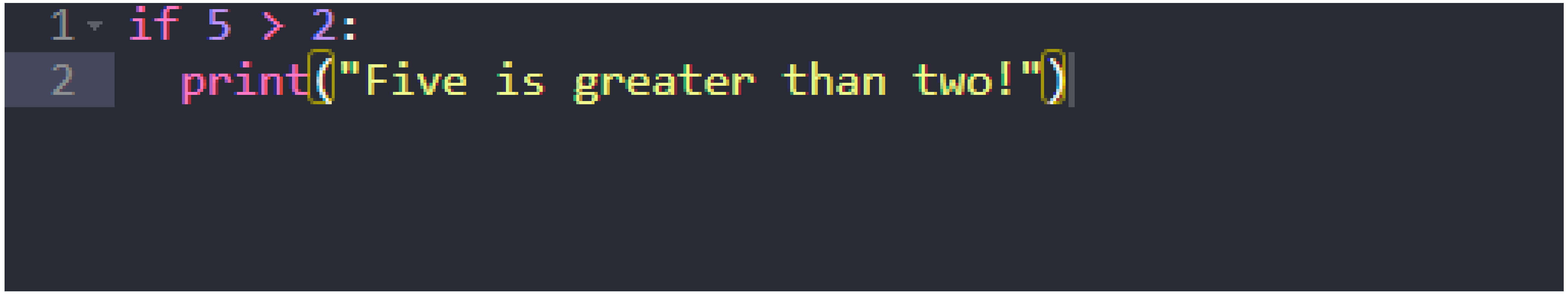
- By default, print() separates the items by spaces.
- The optional sep= parameter sets a different separator text.
- **Example:**
  ```
  a=12
  b=12
  c=2022
  print(a,b,c,sep="-")
  ```
- **Output:**
  ```
  12-12-2022
  ```

What is Comment in Python?

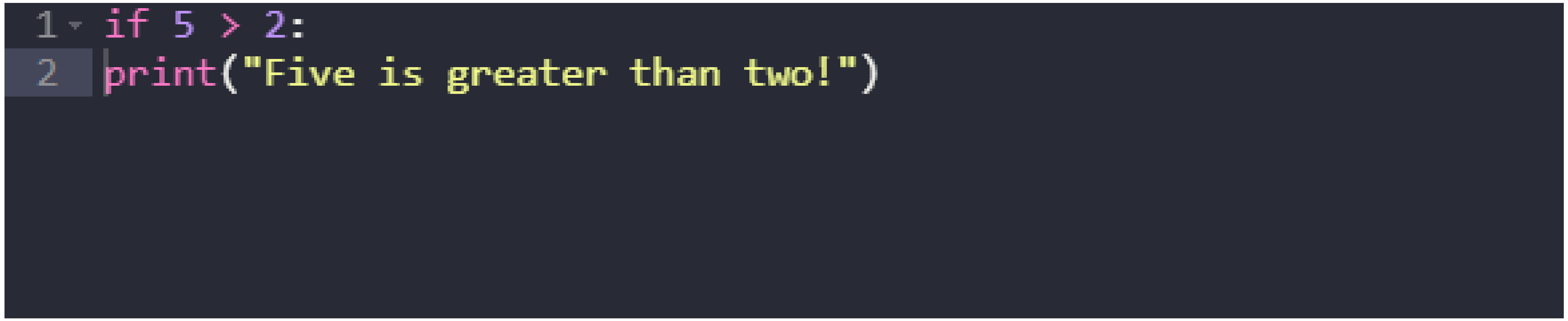
- The comments are descriptions that help programmers to understand the functionality of the program. Thus, comments are necessary while writing code in Python.
- In Python, we **use the hash (#) symbol to start writing a comment**. The comment begins with a hash sign (#) and whitespace character and continues to the end of the line.
- Many programmers commonly use Python for task automation, data analysis, and data visualization. Also, Python has been adopted by many non-programmers such as analysts and scientists.

Python Indentation

- Python indentation is a way of telling a Python interpreter that the group of statements belongs to a particular block of code.
- A block is a combination of all these statements. Block can be regarded as the grouping of statements for a specific purpose.
- Most programming languages like C, C++, and Java use braces { } to define a block of code.
- Python uses indentation to highlight the blocks of code. Whitespace is used for indentation in Python.
- All statements with the same distance to the right belong to the same block of code.
- If a block has to be more deeply nested, it is simply indented further to the right. You can understand it better by looking at the following lines of code.

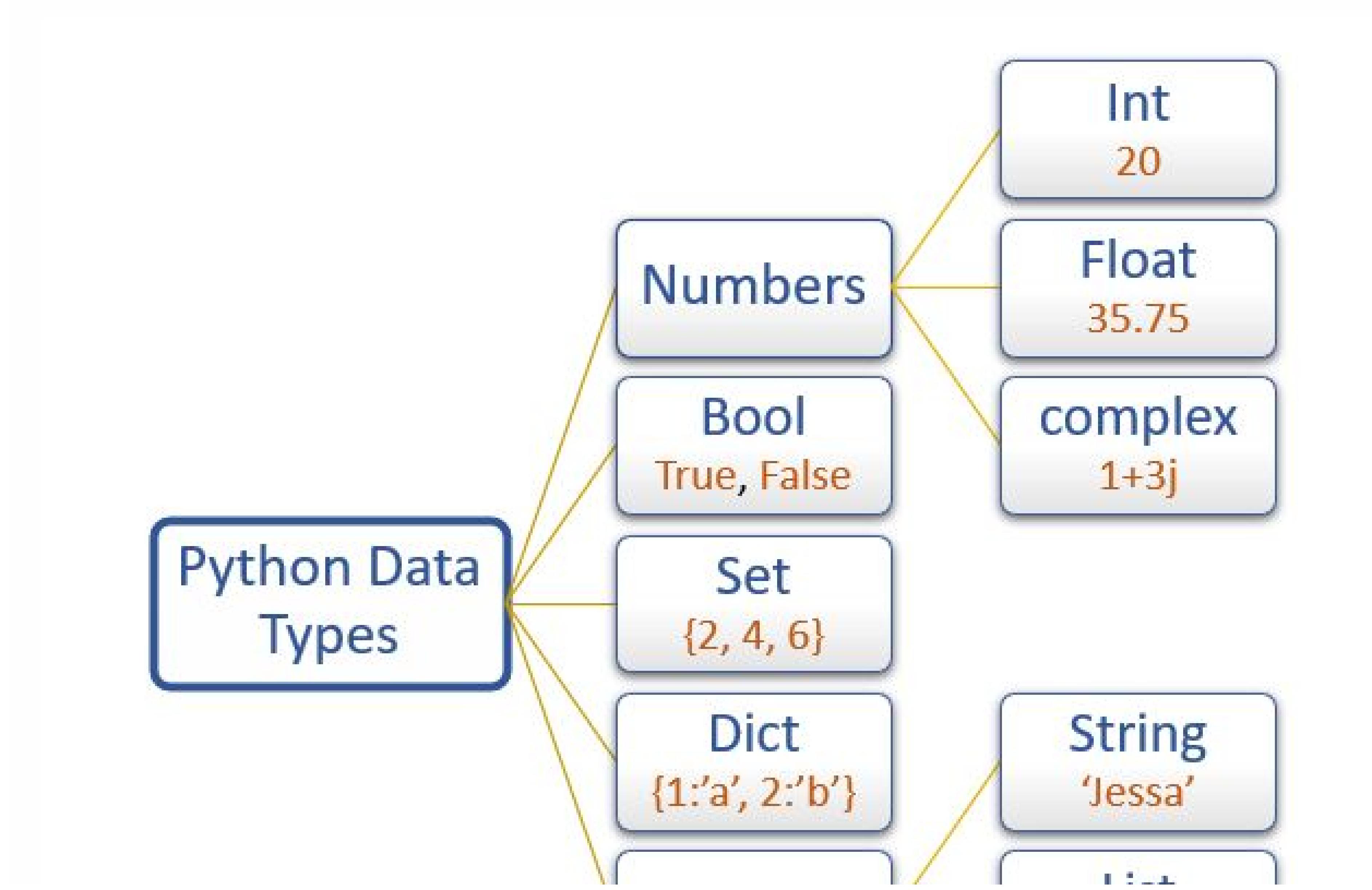


```
if 5 > 2:
    print("Five is greater than two!")
```



```
if 5 > 2:
print("Five is greater than two!")
```

Data Types in Python



Type() function

- To check the data type of variable use the built-in function type()
- The type() function returns the data type of the variable

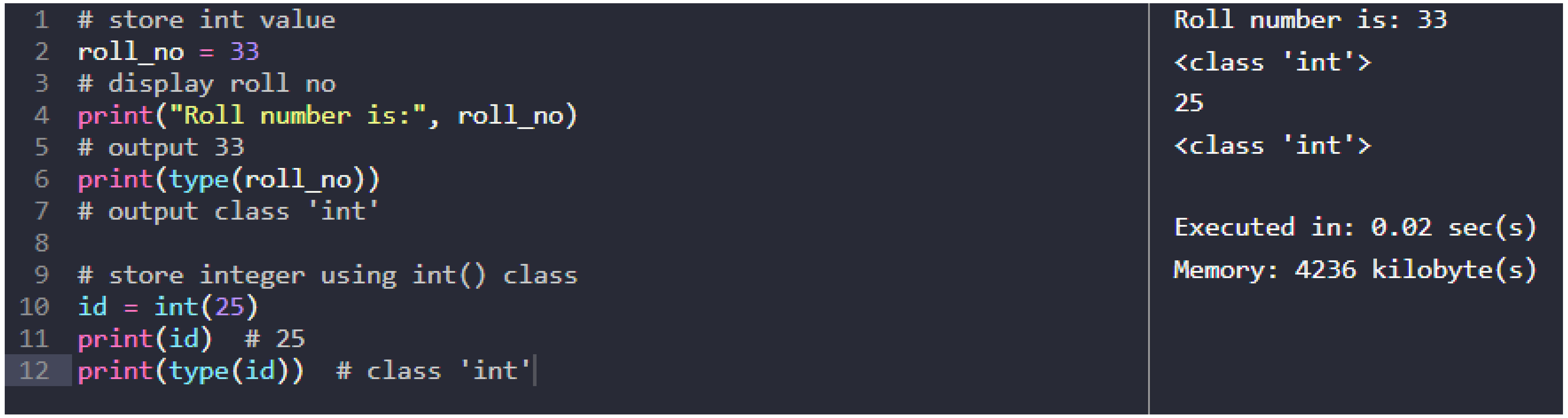
Str data type

- In Python, A string is a **sequence of characters enclosed within a single quote or double quote**. These characters could be anything like letters, numbers, or special symbols enclosed within double quotation marks. For example, "PYnative" is a string.
- The string type in Python is represented using a str class.
- To work with text or character data in Python, we use Strings. Once a string is created, we can do many operations on it, such as searching inside it, creating a substring from it, and splitting it.
- **Example:**

```
form = "native“
print(type(form))
print(form)
Output:
<class 'str'>
 native
```

Int data type

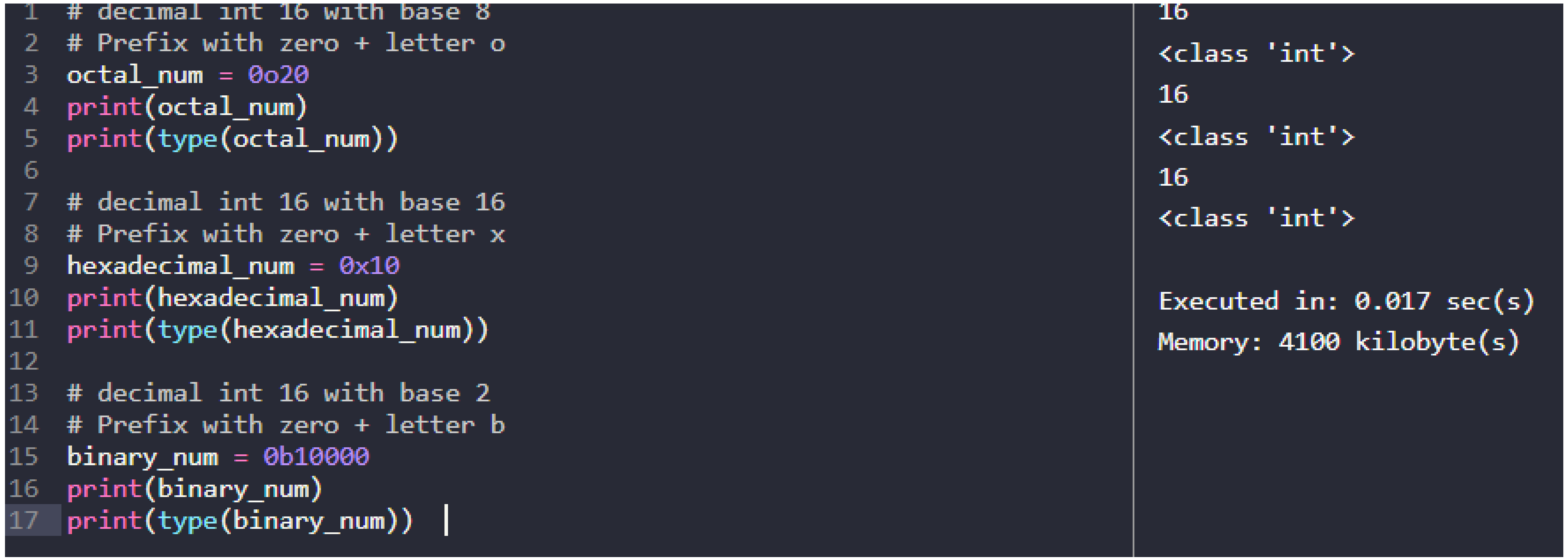
- Python uses the int data type to **represent whole integer values**. For example, we can use the int data type to store the roll number of a student. The Integer type in Python is represented using a int class.
- We can create an integer variable using the two ways:
  1. Directly assigning an integer value to a variable
  2. Using a int() class.



```python
# store int value
roll_no = 33
# display roll no
print("Roll number is:", roll_no)
# output 33
print(type(roll_no))
# output class 'int'

# store integer using int() class
id = int(25)
print(id)  # 25
print(type(id))  # class 'int'
```

```
Roll number is: 33
<class 'int'>
25
<class 'int'>

Executed in: 0.02 sec(s)
Memory: 4236 kilobyte(s)
```

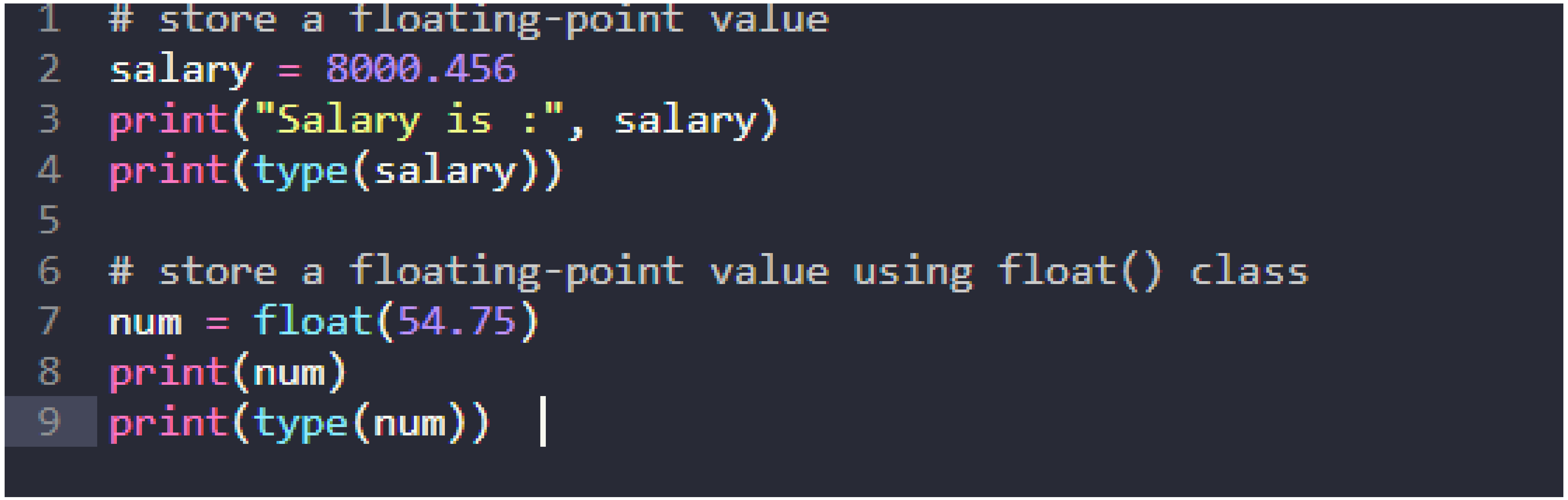
- You can also store integer values other than base 10 such as
- Binary (base 2)
- Octal (base 8)
- Hexadecimal numbers (base 16)


```python
# decimal int 16 with base 8
# Prefix with zero + letter o
octal_num = 0o20
print(octal_num)
print(type(octal_num))

# decimal int 16 with base 16
# Prefix with zero + letter x
hexadecimal_num = 0x10
print(hexadecimal_num)
print(type(hexadecimal_num))

# decimal int 16 with base 2
# Prefix with zero + letter b
binary_num = 0b10000
print(binary_num)
print(type(binary_num))
```

```
16
<class 'int'>
16
<class 'int'>
16
<class 'int'>

Executed in: 0.017 sec(s)
Memory: 4100 kilobyte(s)
```

Float data type

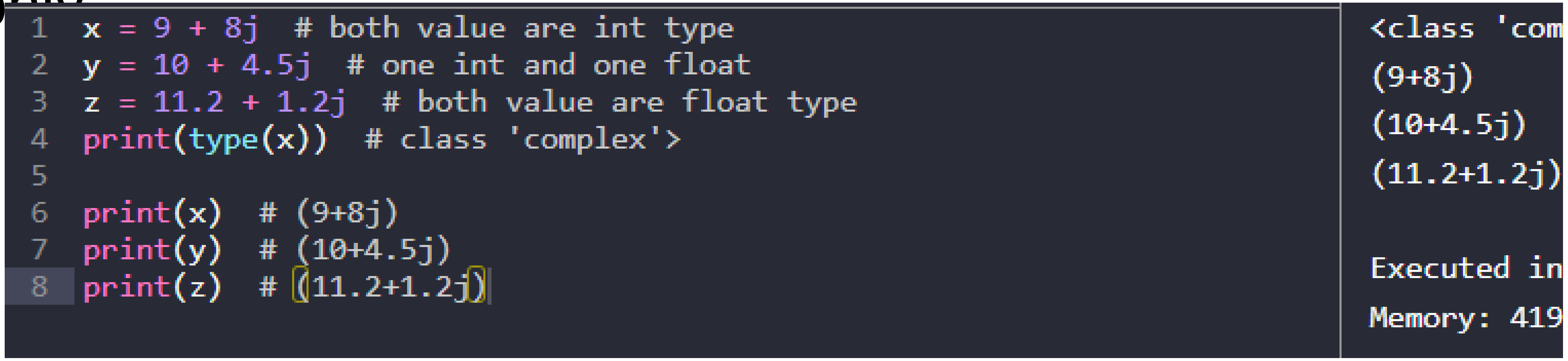
- To represent **floating-point values or decimal value**s, we can use the float data type. For example, if we want to store the salary, we can use the float type.
- The float type in Python is represented using a float class.
- We can create a float variable using the two ways
  1. Directly assigning a float value to a variable
  2. Using a float() class.



```python
# store a floating-point value
salary = 8000.456
print("Salary is :", salary)
print(type(salary))

# store a floating-point value using float() class
num = float(54.75)
print(num)
print(type(num))
```

Complex data type

- A complex number is a **number with a real and an imaginary component** represented as a+bj where a and b contain integers or floating-point values.
- The complex type is generally used in scientific applications and electrical engineering applications. If we want to declare a complex value, then we can use the a+bj form. See the following example.
- **The imaginary part** should be represented using **the decimal** form only.

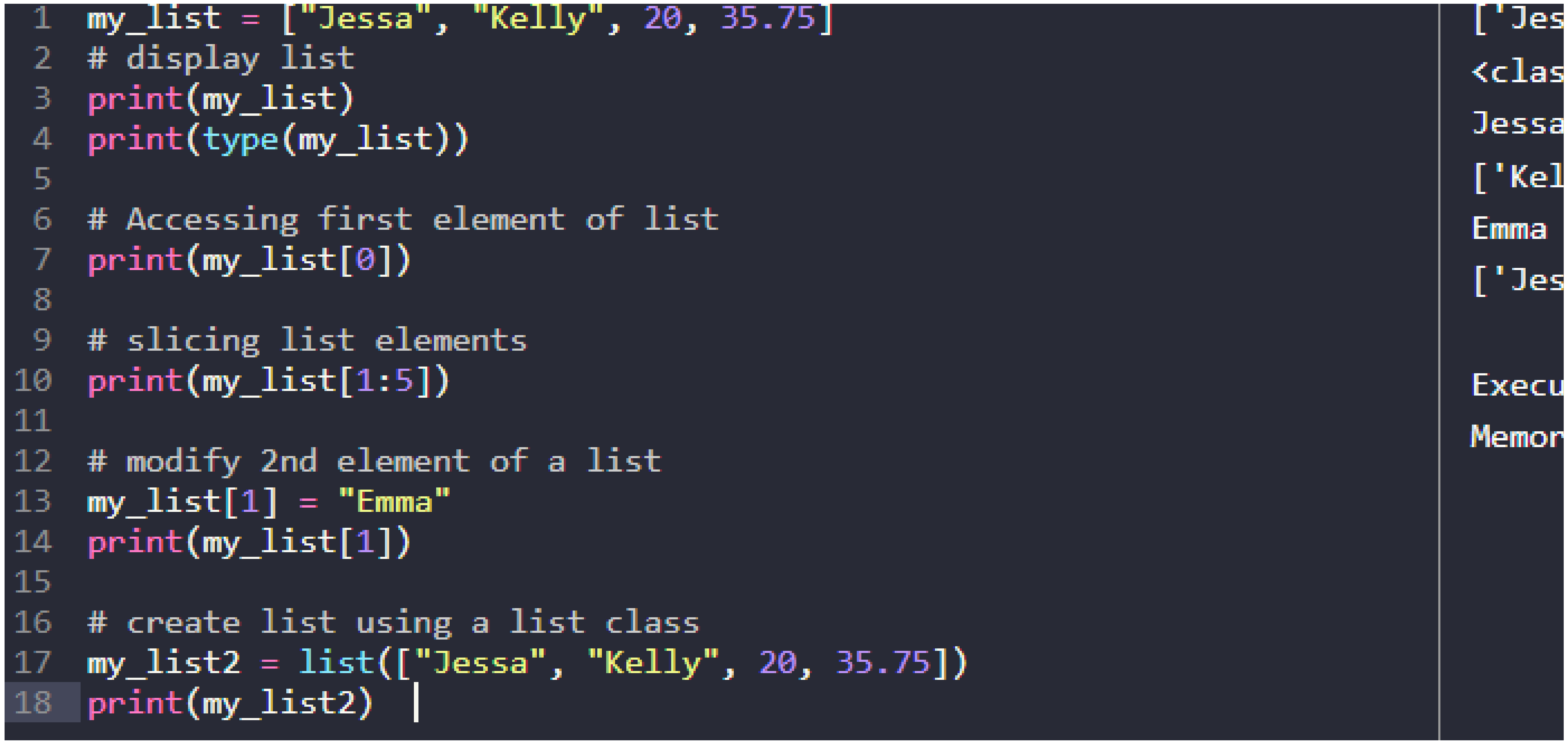


```
x = 9 + 8j  # both value are int type
y = 10 + 4.5j  # one int and one float
z = 11.2 + 1.2j  # both value are float type
print(type(x))  # class 'complex'>

print(x)  # (9+8j)
print(y)  # (10+4.5j)
print(z)  # (11.2+1.2j)
```

```
<class 'com
(9+8j)
(10+4.5j)
(11.2+1.2j)

Executed in
Memory: 419
```

List data type

- The Python List is an **ordered collection (also known as a sequence) of elements**. List elements can be accessed, iterated, and removed according to the order they inserted at the creation time.
- We use the list data type to represent groups of the element as a single entity. For example: If we want to store all student’s names, we can use list type.
- The list can contain data of all data types such as int, float, string
- Duplicates elements are allowed in the list
- The list is mutable which means we can modify the value of list elements
- We can create a list using the two ways:
  1. By enclosing elements in the **square brackets []**.
  2. Using a list() class.

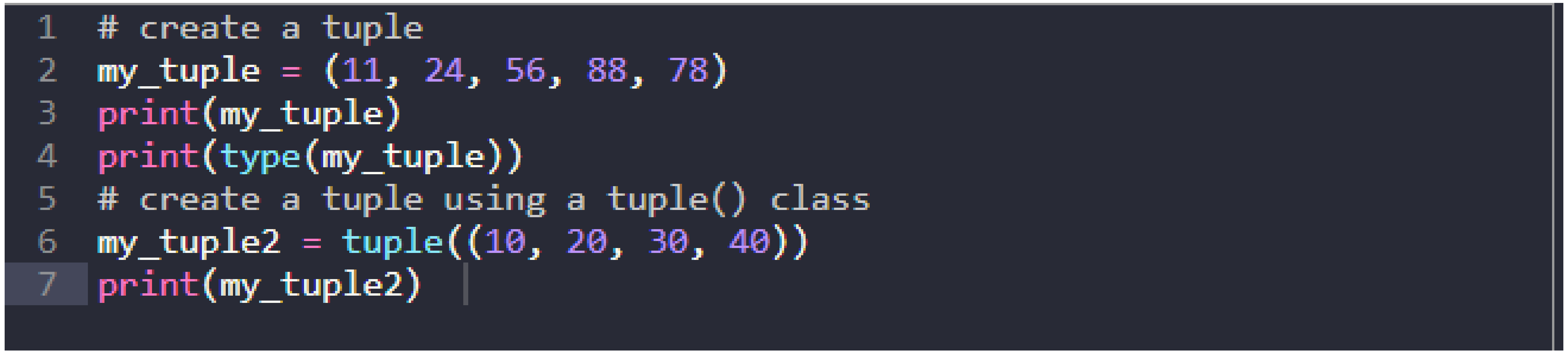


```
my_list = ["Jessa", "Kelly", 20, 35.75]
# display list
print(my_list)
print(type(my_list))

# Accessing first element of list
print(my_list[0])

# slicing list elements
print(my_list[1:5])

# modify 2nd element of a list
my_list[1] = "Emma"
print(my_list[1])

# create list using a list class
my_list2 = list(["Jessa", "Kelly", 20, 35.75])
print(my_list2)
```

```
['Jes
<clas
Jessa
['Kel
Emma
['Jes


Execu
Memor
```

Tuple data type

- Tuples are **ordered collections of elements that are unchangeab**le. The tuple is the same as the list, except the tuple is immutable means we can’t modify the tuple once created.
- In other words, we can say a tuple is a read-only version of the list.
- For example: If you want to store the roll numbers of students that you don’t change, you can use the tuple data type.
- **Note**: Tuple maintains the insertion order and also, allows us to store duplicate elements.
- We can create a tuple using the two ways:
  1. By enclosing elements in the parenthesis ()
  2. Using a tuple() class.

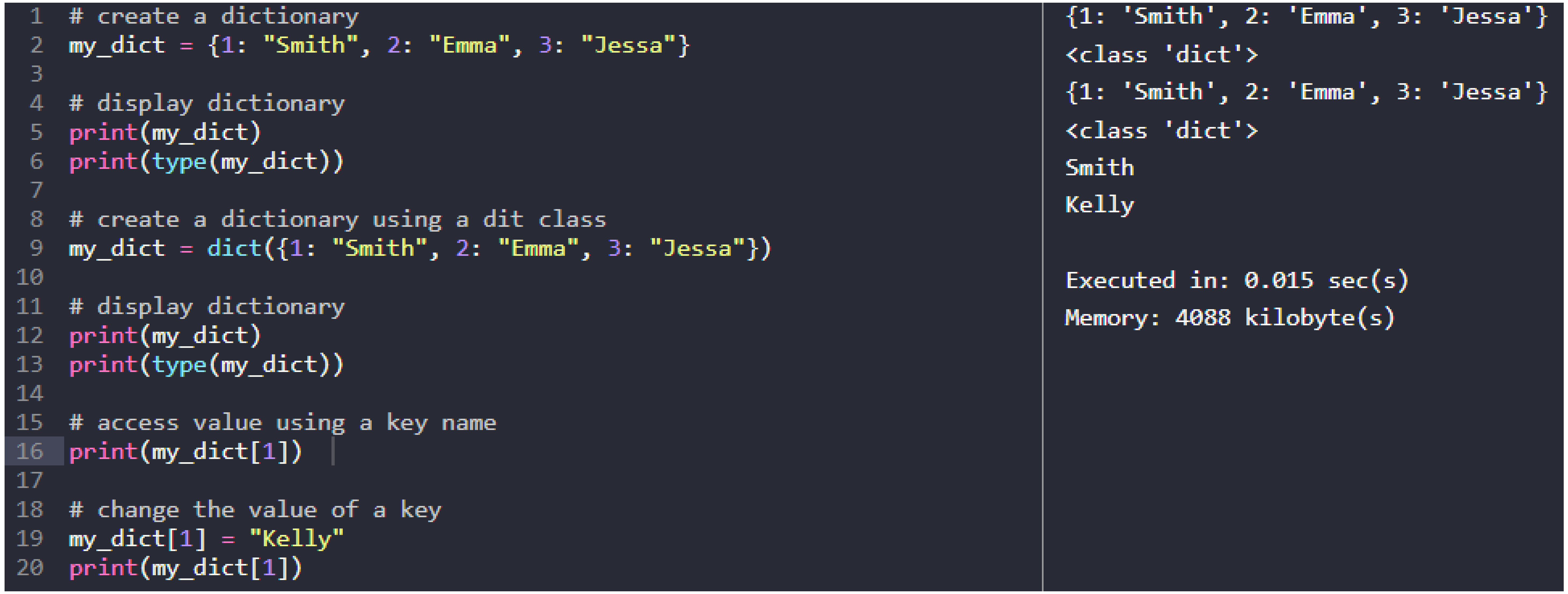


```python
# create a tuple
my_tuple = (11, 24, 56, 88, 78)
print(my_tuple)
print(type(my_tuple))
# create a tuple using a tuple() class
my_tuple2 = tuple((10, 20, 30, 40))
print(my_tuple2)
```

Dict data type

- In Python, dictionaries are **unordered collections of unique values stored in (Key-Value) pairs**. Use a dictionary data type to store data as a key-value pair.
- The dictionary type is represented using a dict class. For example, If you want to store the name and roll number of all students, then you can use the dict type.
- In a dictionary, duplicate keys are not allowed, but the value can be duplicated. If we try to insert a value with a duplicate key, the old value will be replaced with the new value.
- Dictionary has some characteristics which are listed below:
- A heterogeneous (i.e., str, list, tuple) elements are allowed for both key and value in a dictionary. But An object can be a key in a dictionary if it is hashable.
- The dictionary is mutable which means we can modify its items
- Dictionary is unordered so we can’t perform indexing and slicing

- We can create a dictionary using the two ways
- By enclosing key and values in the curly brackets {}
- Using a dict() class.

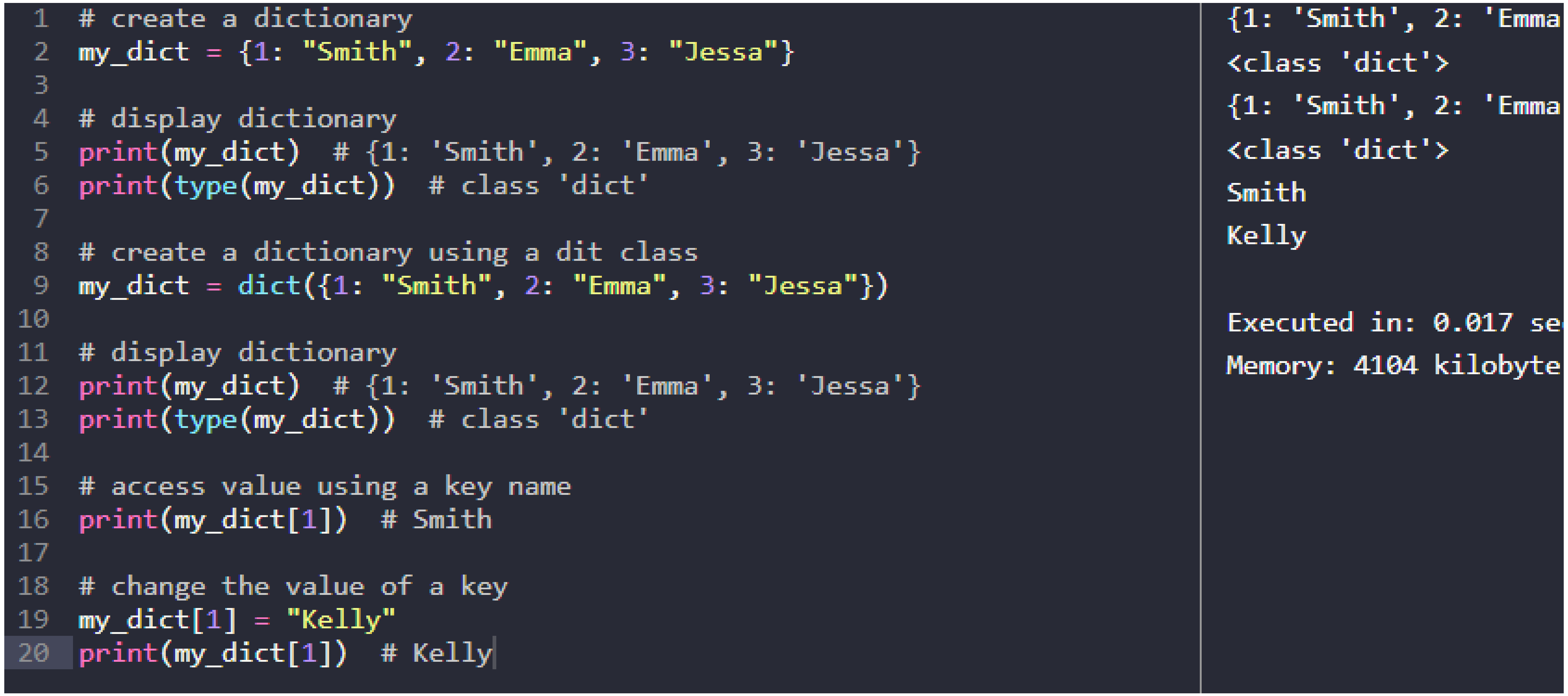


```python
# create a dictionary
my_dict = {1: "Smith", 2: "Emma", 3: "Jessa"}

# display dictionary
print(my_dict)
print(type(my_dict))

# create a dictionary using a dit class
my_dict = dict({1: "Smith", 2: "Emma", 3: "Jessa"})

# display dictionary
print(my_dict)
print(type(my_dict))

# access value using a key name
print(my_dict[1])

# change the value of a key
my_dict[1] = "Kelly"
print(my_dict[1])
```

```
{1: 'Smith', 2: 'Emma', 3: 'Jessa'}
<class 'dict'>
{1: 'Smith', 2: 'Emma', 3: 'Jessa'}
<class 'dict'>
Smith
Kelly

Executed in: 0.015 sec(s)
Memory: 4088 kilobyte(s)
```

Set data type

- In Python, a set is an **unordered collection of data items that are unique**. In other words, Python Set is a collection of elements (Or objects) that contains no duplicate elements.
- In Python, the Set data type used to represent a group of unique elements as a single entity. For example, If we want to store student ID numbers, we can use the set data type.
- The Set data type in Python is represented using a set class.
- We can create a Set using the two ways
- By enclosing values in the curly brackets {}
- Using a set() class.
- **The set data type has the following characteristics.**
- It is mutable which means we can change set items
- Duplicate elements are not allowed
- Heterogeneous (values of all data types) elements are allowed
- Insertion order of elements is not preserved, so we can’t perform indexing on a Set

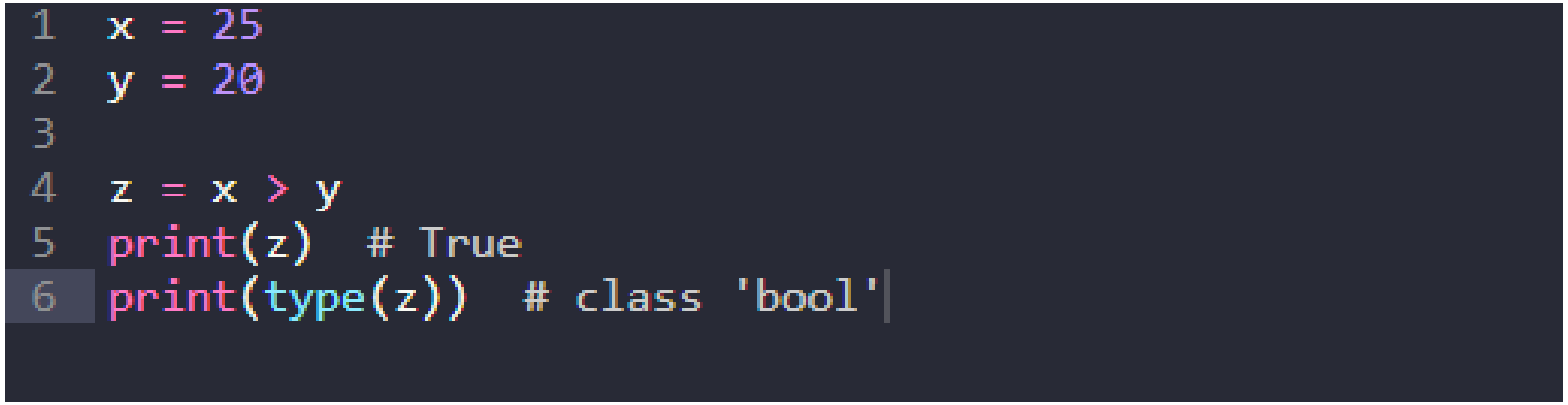


```python
# create a dictionary
my_dict = {1: "Smith", 2: "Emma", 3: "Jessa"}

# display dictionary
print(my_dict)  # {1: 'Smith', 2: 'Emma', 3: 'Jessa'}
print(type(my_dict))  # class 'dict'

# create a dictionary using a dit class
my_dict = dict({1: "Smith", 2: "Emma", 3: "Jessa"})

# display dictionary
print(my_dict)  # {1: 'Smith', 2: 'Emma', 3: 'Jessa'}
print(type(my_dict))  # class 'dict'

# access value using a key name
print(my_dict[1])  # Smith

# change the value of a key
my_dict[1] = "Kelly"
print(my_dict[1])  # Kelly
```

```
{1: 'Smith', 2: 'Emma
<class 'dict'>
{1: 'Smith', 2: 'Emma
<class 'dict'>
Smith
Kelly

Executed in: 0.017 se
Memory: 4104 kilobyte
```

Bool data type

- In Python, to **represent boolean values (True and False)** we use the bool data type. Boolean values are used to evaluate the value of the expression. For example, when we compare two values, the expression is evaluated, and Python returns the boolean True or False.



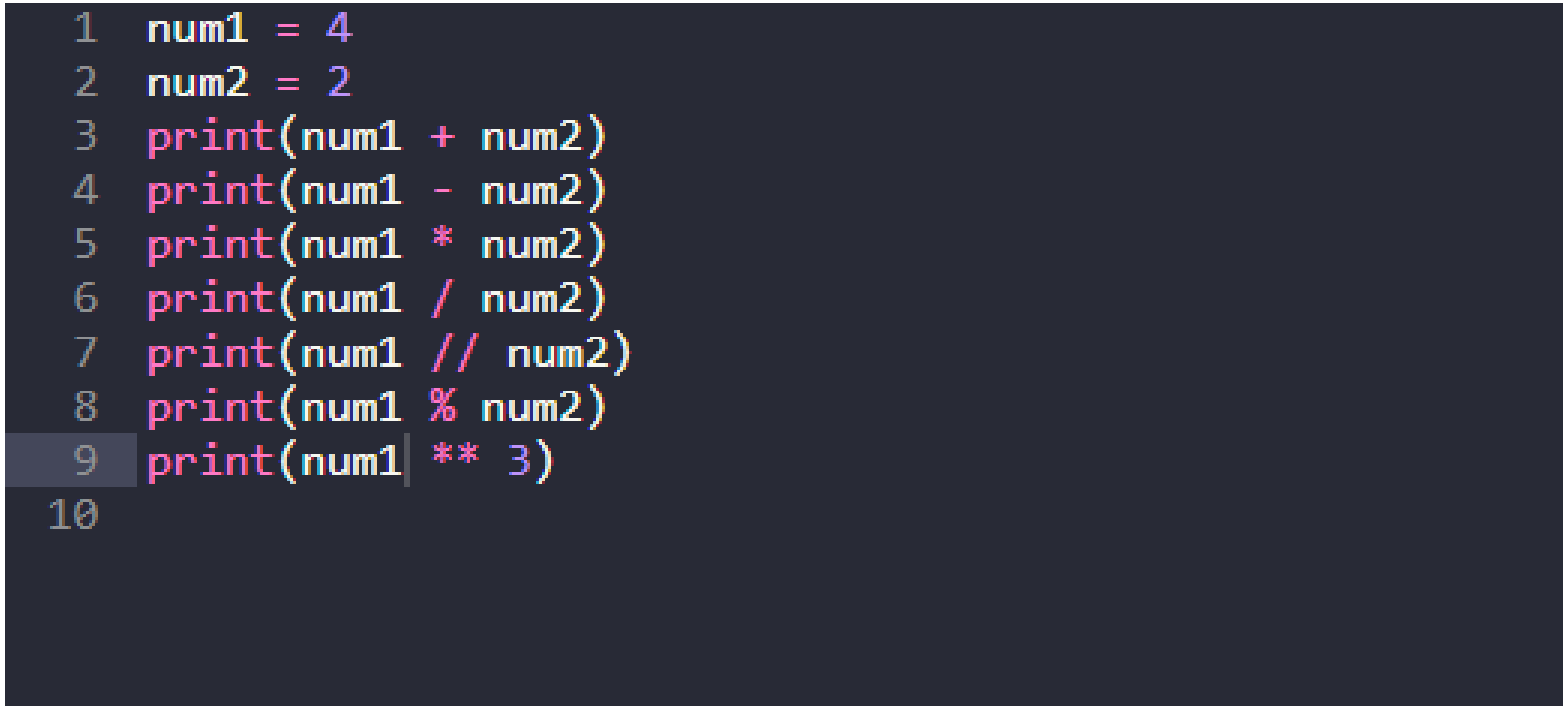
```
x = 25
y = 20

z = x > y
print(z)  # True
print(type(z))  # class 'bool'
```

Operators

- Python has seven types of operators that we can use to perform different operation and produce a result.
- Arithmetic operator
- Relational operators
- Assignment operators
- Logical operators
- Membership operators
- Identity operators
- Bitwise operators

Arithmetic Operators

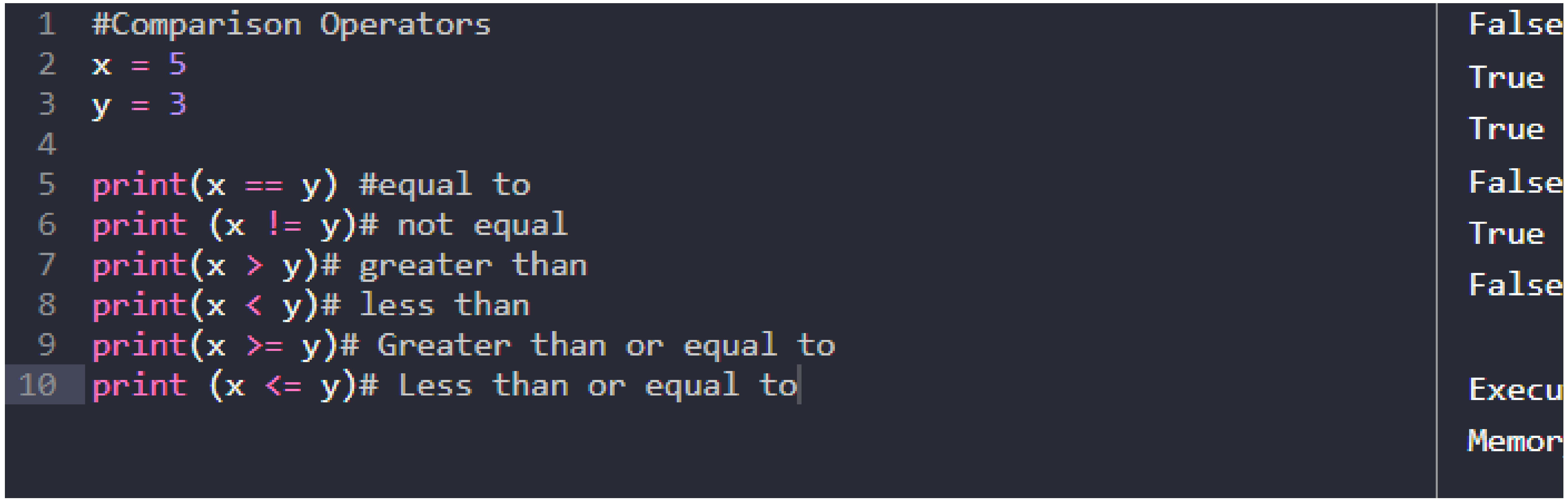
- Arithmetic operators are the most commonly used. The Python programming language provides arithmetic operators that perform addition, subtraction, multiplication, and division. It works the same as basic mathematics.
- There are seven arithmetic operators we can use to perform different mathematical operations, such as:
- + (Addition)
- - (Subtraction)
- * (Multiplication)
- / (Division)
- // (Floor division) : Floor division returns the quotient (the result of division) in which the digits after the decimal point are removed. In simple terms, It is used to divide one value by a second value and gives a quotient as a round figure value to the next smallest whole value.
- % (Modulus): The remainder of the division of left operand by the right. The modulus operator is denoted by a % symbol. In simple terms, the Modulus operator divides one value by a second and gives the remainder as a result.
- ** (Exponentiation) : Using exponent operator left operand raised to the power of right. The exponentiation operator is denoted by a double asterisk ** symbol. You can use it as a shortcut to calculate the exponential value.



```python
num1 = 4
num2 = 2
print(num1 + num2)
print(num1 - num2)
print(num1 * num2)
print(num1 / num2)
print(num1 // num2)
print(num1 % num2)
print(num1 ** 3)
```

Relational (comparison) Operators

- Relational operators are also called comparison operators. It performs a comparison between two values. It returns a boolean True or False depending upon the result of the comparison.
- Python has the following six relational operators.

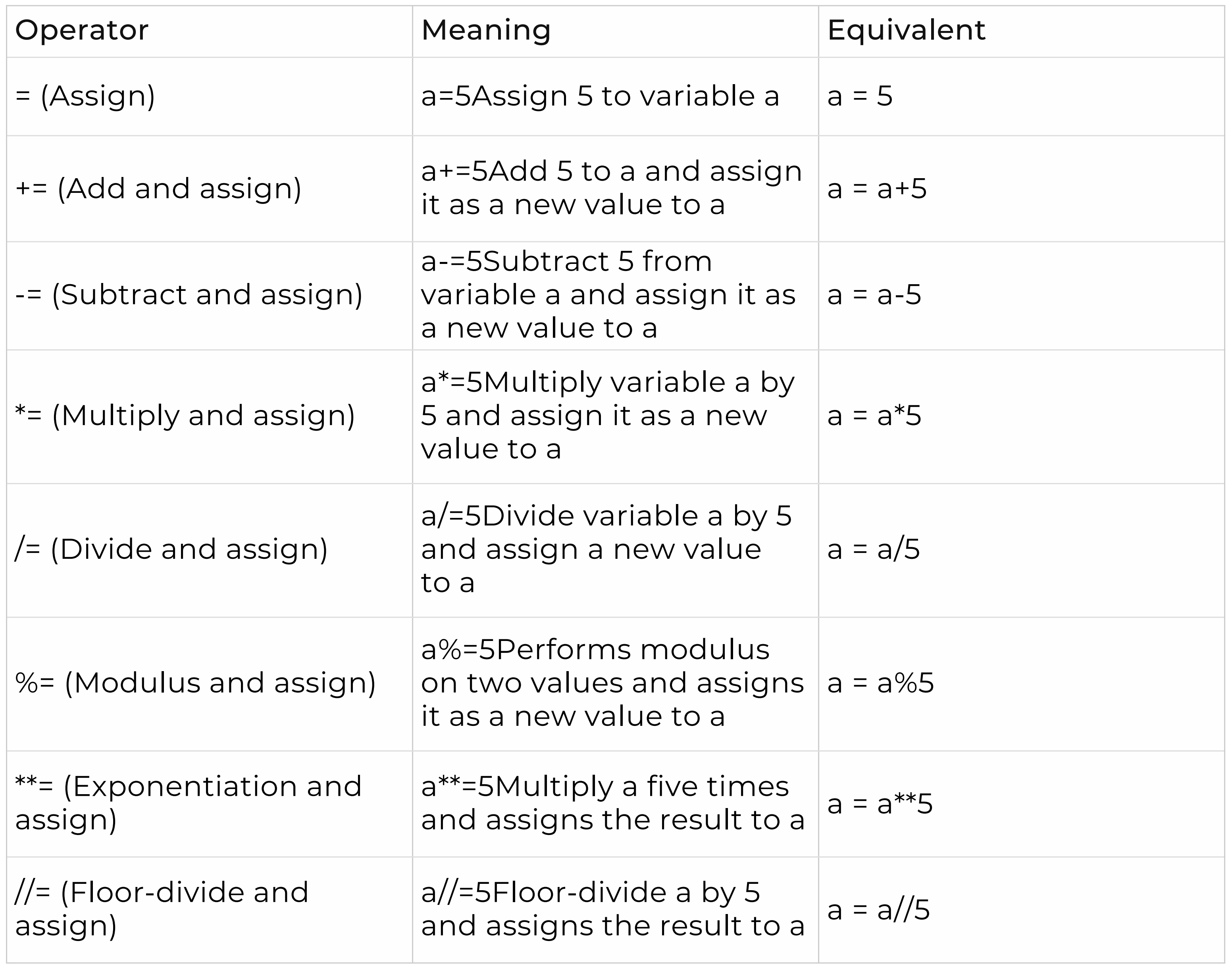


```
#Comparison Operators
x = 5
y = 3

print(x == y) #equal to
print (x != y)# not equal
print(x > y)# greater than
print(x < y)# less than
print(x >= y)# Greater than or equal to
print (x <= y)# Less than or equal to
```

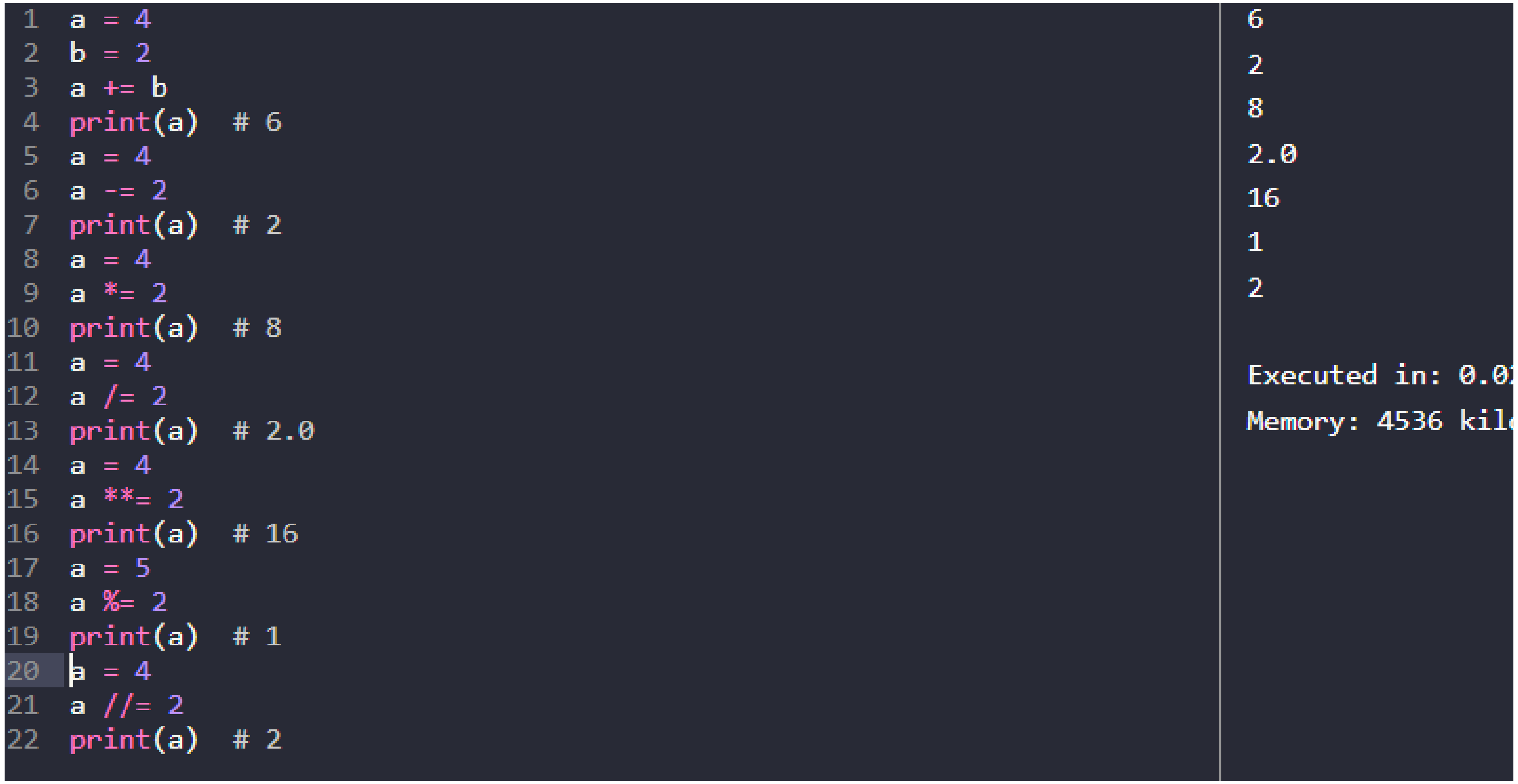
```
False
True
True
False
True
False


Execu
Memor
```

Assignment Operators

- In Python, Assignment operators are used to assigning value to the variable. Assign operator is denoted by = symbol. For example, name = "Jessa" here, we have assigned the string literal ‘Jessa’ to a variable name.



Operator	Meaning	Equivalent
= (Assign)	a=5Assign 5 to variable a	a = 5
+= (Add and assign)	a+=5Add 5 to a and assign it as a new value to a	a = a+5
-= (Subtract and assign)	a-=5Subtract 5 from variable a and assign it as a new value to a	a = a-5
*= (Multiply and assign)	a*=5Multiply variable a by 5 and assign it as a new value to a	a = a*5
/= (Divide and assign)	a/=5Divide variable a by 5 and assign a new value to a	a = a/5
%= (Modulus and assign)	a%=5Performs modulus on two values and assigns it as a new value to a	a = a%5
**= (Exponentiation and assign)	a**=5Multiply a five times and assigns the result to a	a = a**5
//= (Floor-divide and assign)	a//=5Floor-divide a by 5 and assigns the result to a	a = a//5



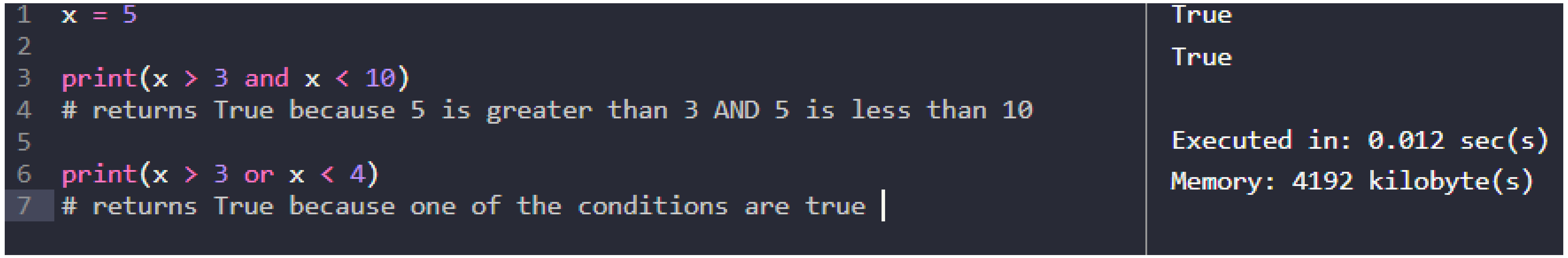
```python
a = 4
b = 2
a += b
print(a)  # 6
a = 4
a -= 2
print(a)  # 2
a = 4
a *= 2
print(a)  # 8
a = 4
a /= 2
print(a)  # 2.0
a = 4
a **= 2
print(a)  # 16
a = 5
a %= 2
print(a)  # 1
a = 4
a //= 2
print(a)  # 2
```

```
6
2
8
2.0
16
1
2

Executed in: 0.0
Memory: 4536 kil
```

Logical Operators

- Logical operators are useful when checking a condition is true or not. Python has three logical operators. All logical operator returns a boolean value True or False depending on the condition in which it is used.

Operator	Description	Example
and (Logical and)	True if both the operands are True	a and b
or (Logical or)	True if either of the operands is True	a or b
not (Logical not)	True if the operand is False	not a

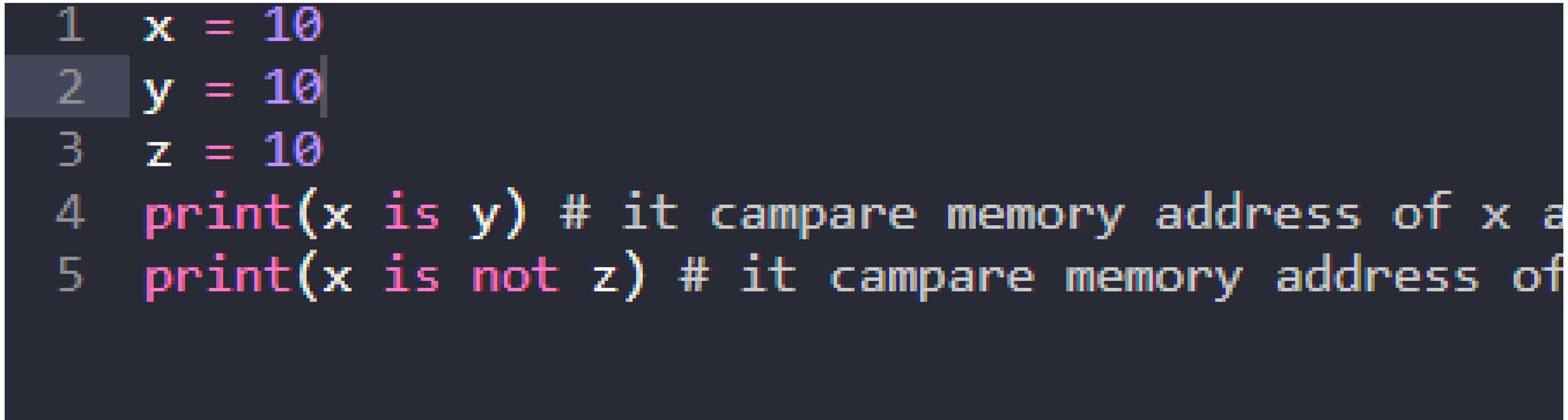


```python
x = 5

print(x > 3 and x < 10)
# returns True because 5 is greater than 3 AND 5 is less than 10

print(x > 3 or x < 4)
# returns True because one of the conditions are true
```

```
True
True

Executed in: 0.012 sec(s)
Memory: 4192 kilobyte(s)
```

Identity Operators

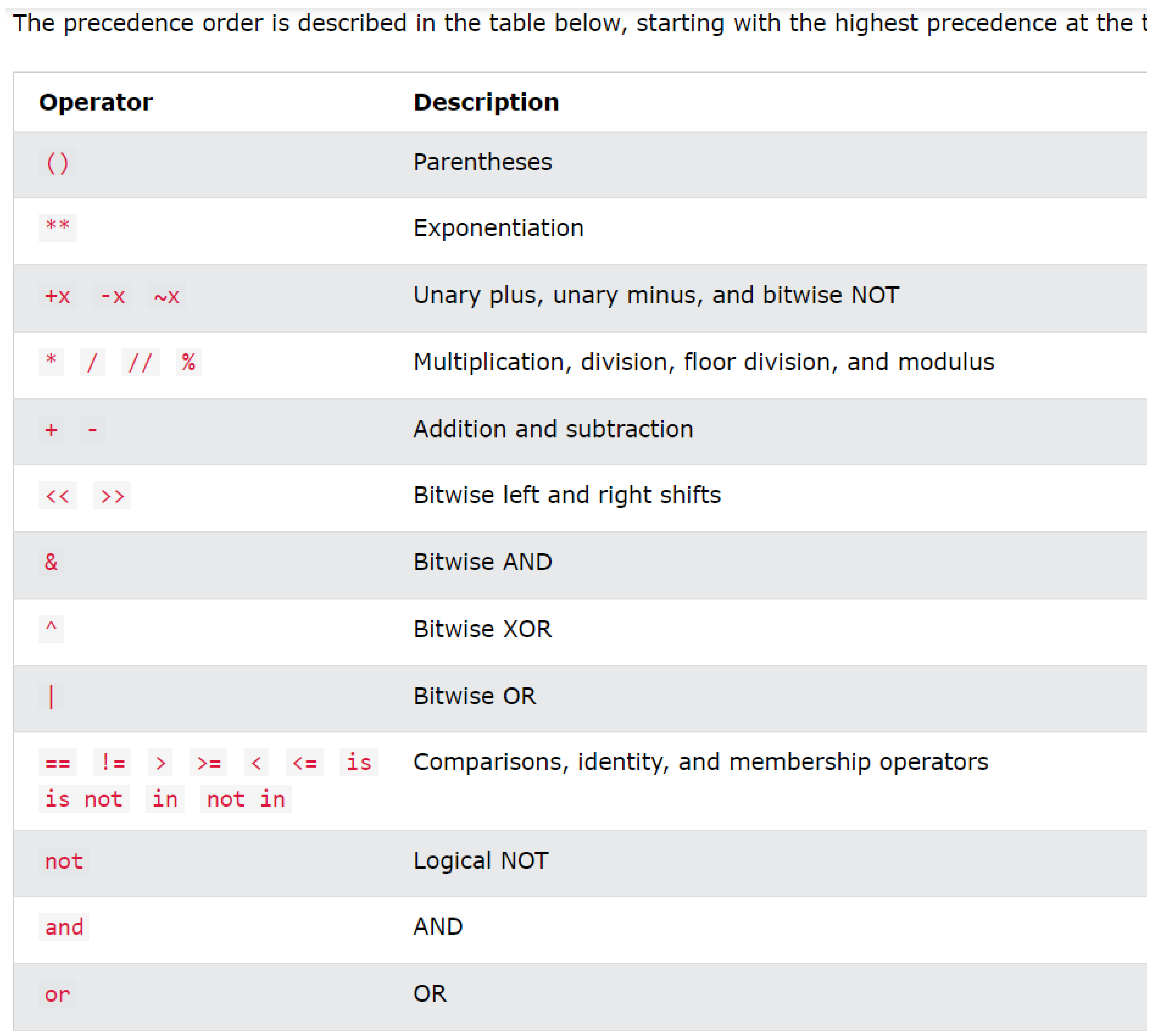
- Use the Identity operator to check whether the value of two variables is the same or not. This operator is known as a **reference-quality operator** because the identity operator compares values according to two variables’ memory addresses.
- Python has 2 identity operators is and is not.
- **is operator**
- The is operator returns Boolean True or False. It Return True if the memory address first value is equal to the second value. Otherwise, it returns False.
- **is not operator**
- The is not the operator returns boolean values either True or False. It Return True if the first value is not equal to the second value. Otherwise, it returns False.



```
x = 10
y = 10
z = 10
print(x is y) # it campare memory address of x a
print(x is not z) # it campare memory address of
```

Bitwise Operators

- In Python, bitwise operators are used to performing bitwise operations on integers. To perform bitwise, we first need to convert integer value to binary (0 and 1) value.
- The bitwise operator operates on values bit by bit, so it’s called **bitwise**. It always returns the result in decimal format. Python has 6 bitwise operators listed below.
- & Bitwise and
- | Bitwise or
- ^ Bitwise xor
- ~ Bitwise 1’s complement
- << Bitwise left-shift
- >> Bitwise right-shift



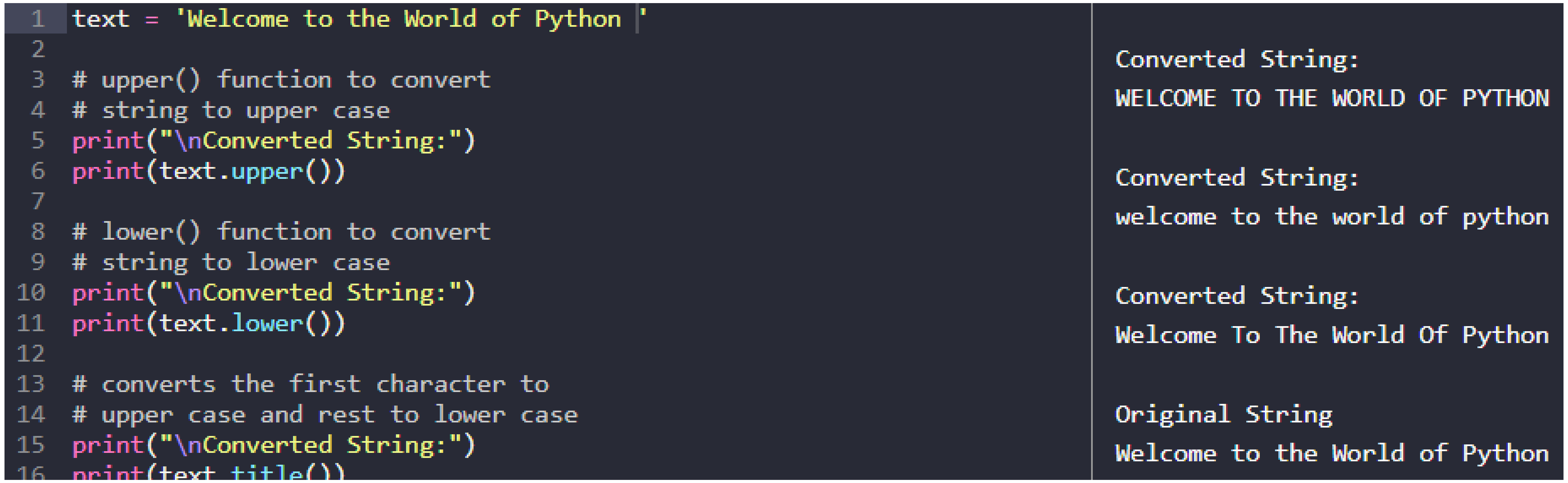
The precedence order is described in the table below, starting with the highest precedence at the t

Operator	Description
`()`	Parentheses
`**`	Exponentiation
`+x` `-x` `~x`	Unary plus, unary minus, and bitwise NOT
`*` `/` `//` `%`	Multiplication, division, floor division, and modulus
`+` `-`	Addition and subtraction
`<<` `>>`	Bitwise left and right shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==` `!=` `>` `>=` `<` `<=` `is` `is not` `in` `not in`	Comparisons, identity, and membership operators
`not`	Logical NOT
`and`	AND
`or`	OR

Operations on Strings

1. Case Changing of Strings

- The below functions are used to change the case of the strings.
- **lower():** Converts all uppercase characters in a string into lowercase
- **upper():** Converts all lowercase characters in a string into uppercase
- **title():** Convert string to title case



```python
text = 'Welcome to the World of Python '

# upper() function to convert
# string to upper case
print("\nConverted String:")
print(text.upper())

# lower() function to convert
# string to lower case
print("\nConverted String:")
print(text.lower())

# converts the first character to
# upper case and rest to lower case
print("\nConverted String:")
print(text.title())
```

```

Converted String:
WELCOME TO THE WORLD OF PYTHON

Converted String:
welcome to the world of python

Converted String:
Welcome To The World Of Python

Original String
Welcome to the World of Python
```

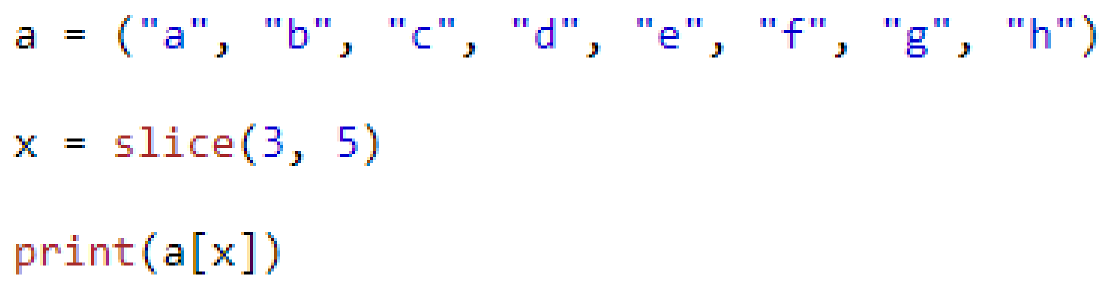
2. Slicing of Strings

- You can extract subsets of strings by using the slice operator.
- syntax

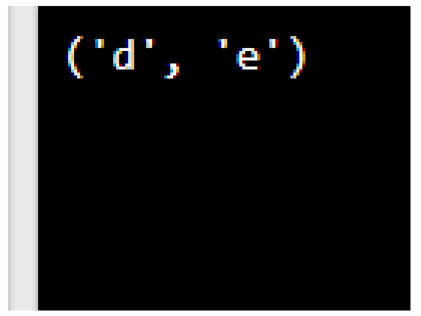
  slice(*start*, *end*, *step*)

  Parameter Values

Parameter	Description
start	Optional. An integer number specifying at which position to start the slicing. Default is 0
end	An integer number specifying at which position to end the slicing
step	Optional. An integer number specifying the step of the slicing. Default is 1



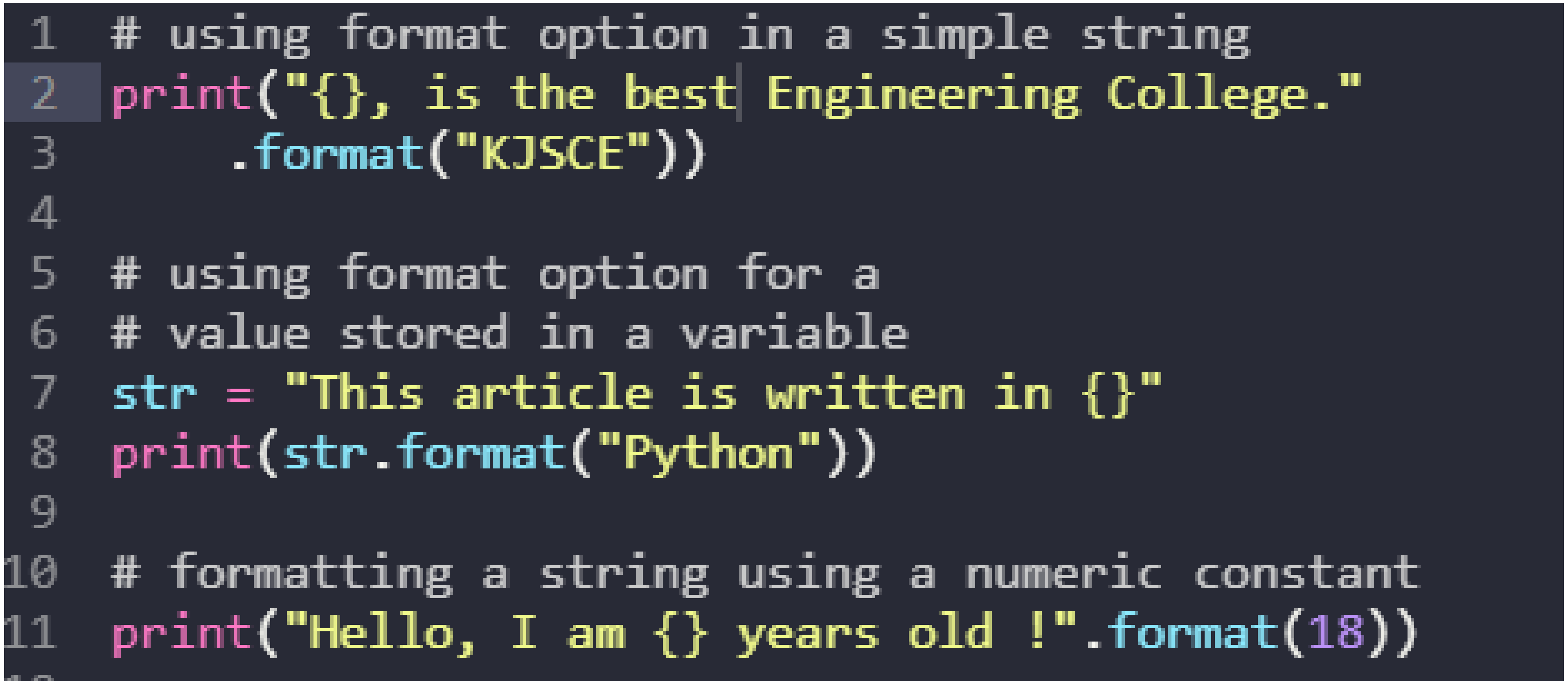
```
a = ("a", "b", "c", "d", "e", "f", "g", "h")

x = slice(3, 5)

print(a[x])
```



```
('d', 'e')
```

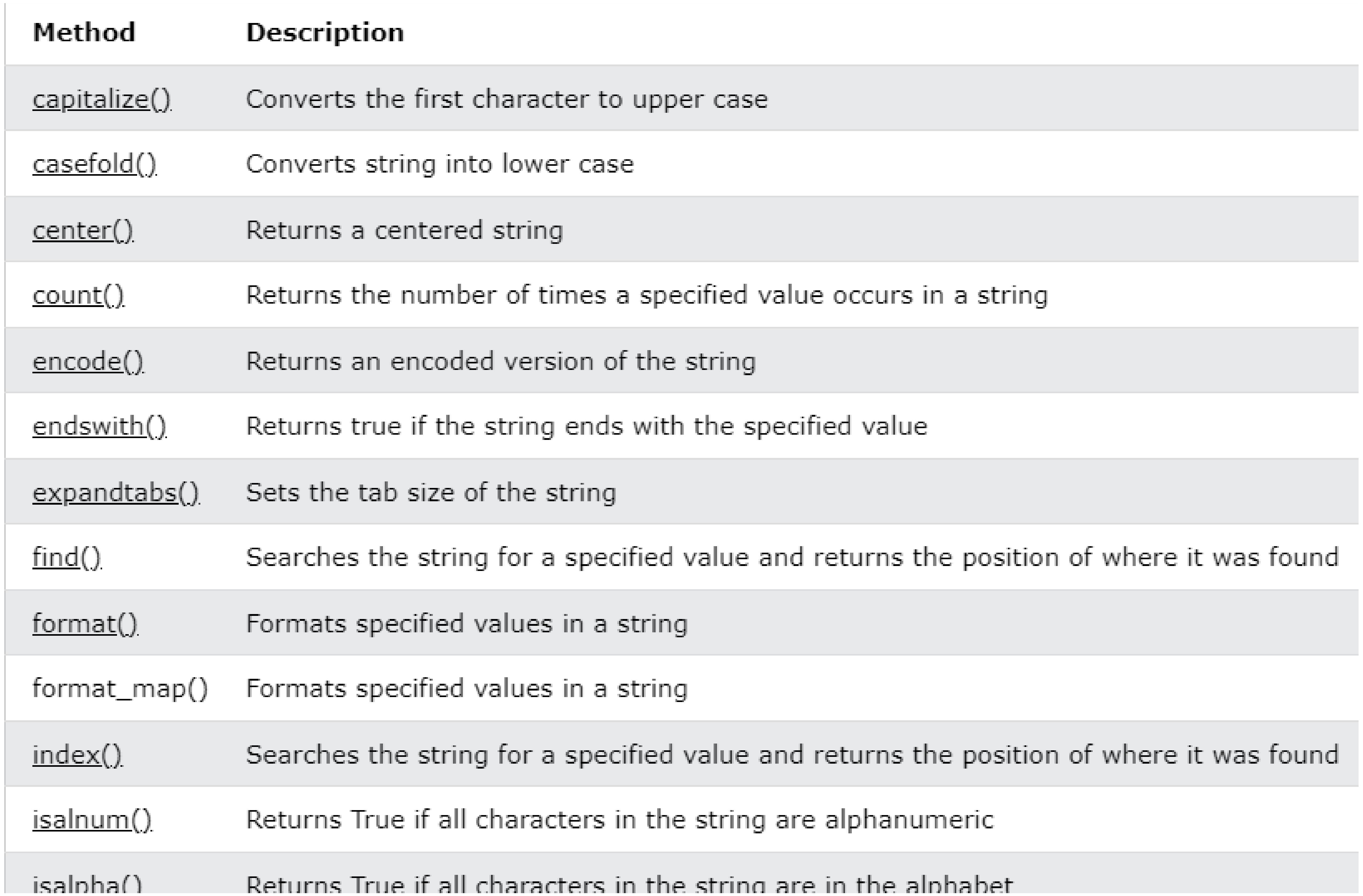
String format() Function

- **format() function** has been introduced for handling complex string formatting more efficiently.
- **A simple demonstration of Python String format() Method**
- Formatters work by putting in one or more replacement fields and placeholders defined by a pair of curly braces **{ }** into a string and calling the format() function. The value we wish to put into the placeholders and concatenate with the string passed as parameters into the format function.
- ***Syntax:***
  { }.format(value)
- ***Parameters:***
  value : *Can be an integer, floating point numeric constant, string, characters or even variables.*
  Returntype: *Returns a formatted string with the value passed as parameter in the placeholder position.*

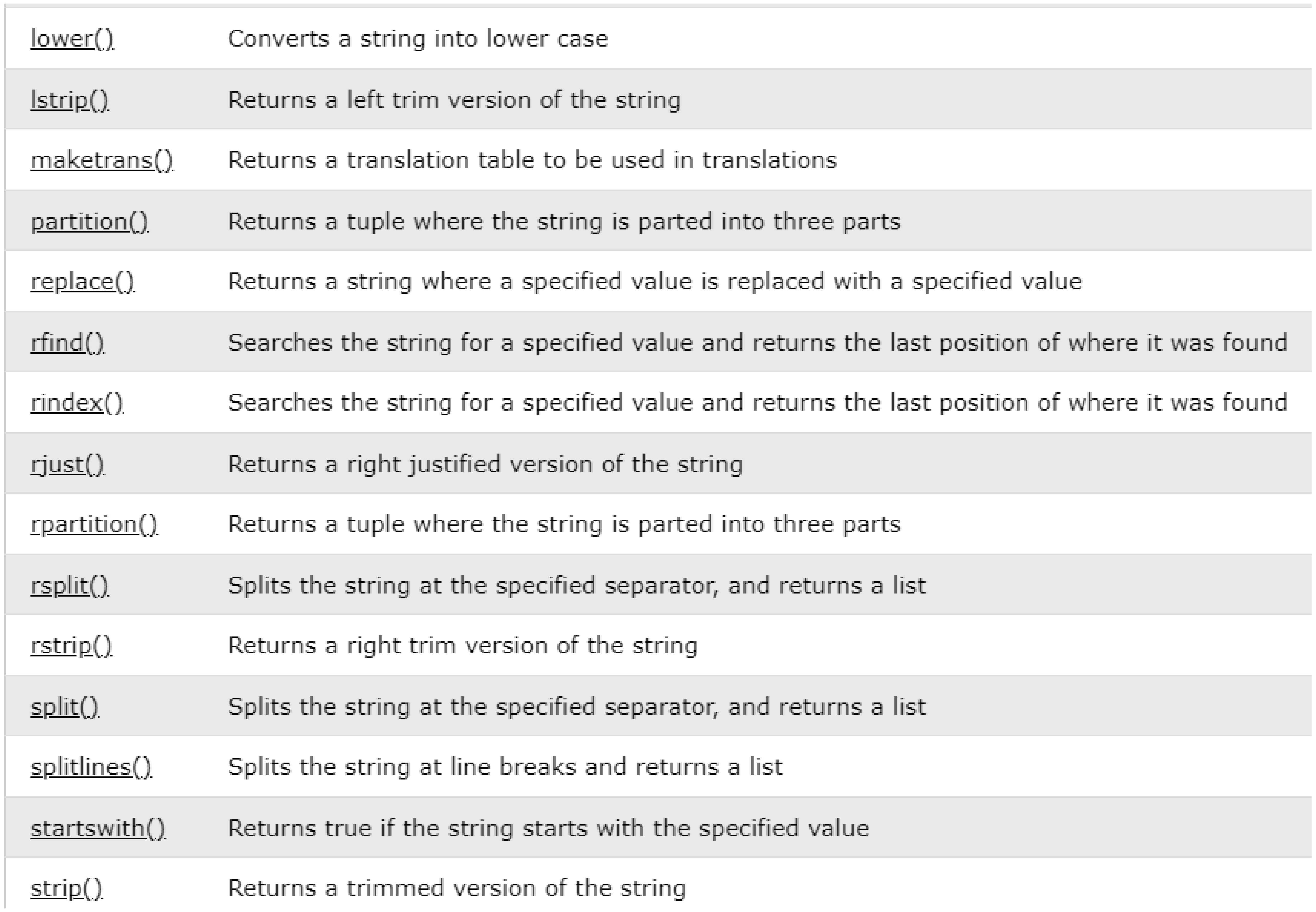


```python
# using format option in a simple string
print("{}, is the best Engineering College."
    .format("KJSCE"))

# using format option for a
# value stored in a variable
str = "This article is written in {}"
print(str.format("Python"))

# formatting a string using a numeric constant
print("Hello, I am {} years old !".format(18))
```

String Methods



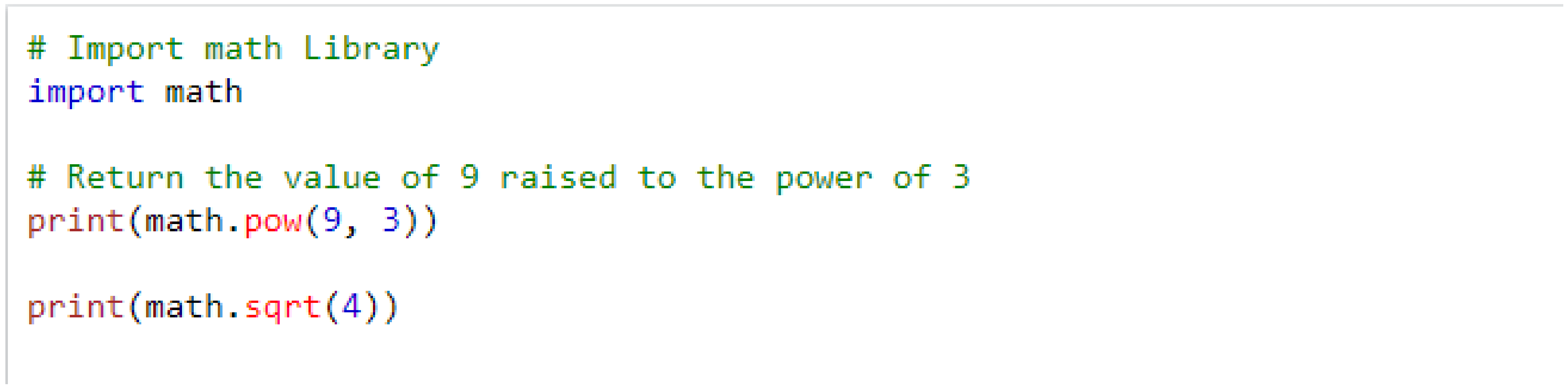
Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet



lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string

math module in Python

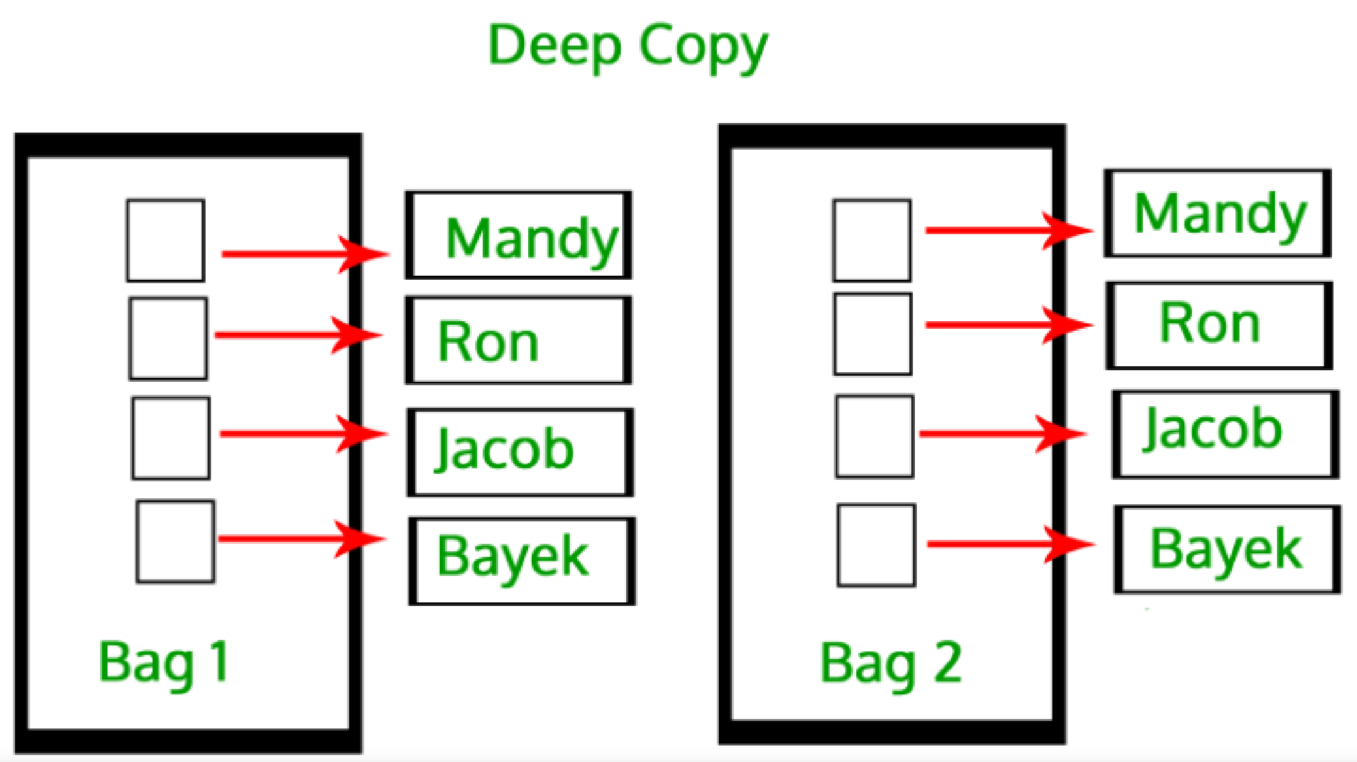
- The math module is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using import math.
- It gives access to the underlying C library functions. For example,
  Example:



```
# Import math Library
import math

# Return the value of 9 raised to the power of 3
print(math.pow(9, 3))

print(math.sqrt(4))
```

What is Deep copy in Python?

A deep copy creates a new compound object before inserting copies of the items found in the original into it in a recursive manner. It means first constructing a new collection object and then recursively populating it with copies of the child objects found in the original. In the case of deep copy, a copy of the object is copied into another object. It means that **any changes** made to a copy of the object **do not reflect in the original object.**

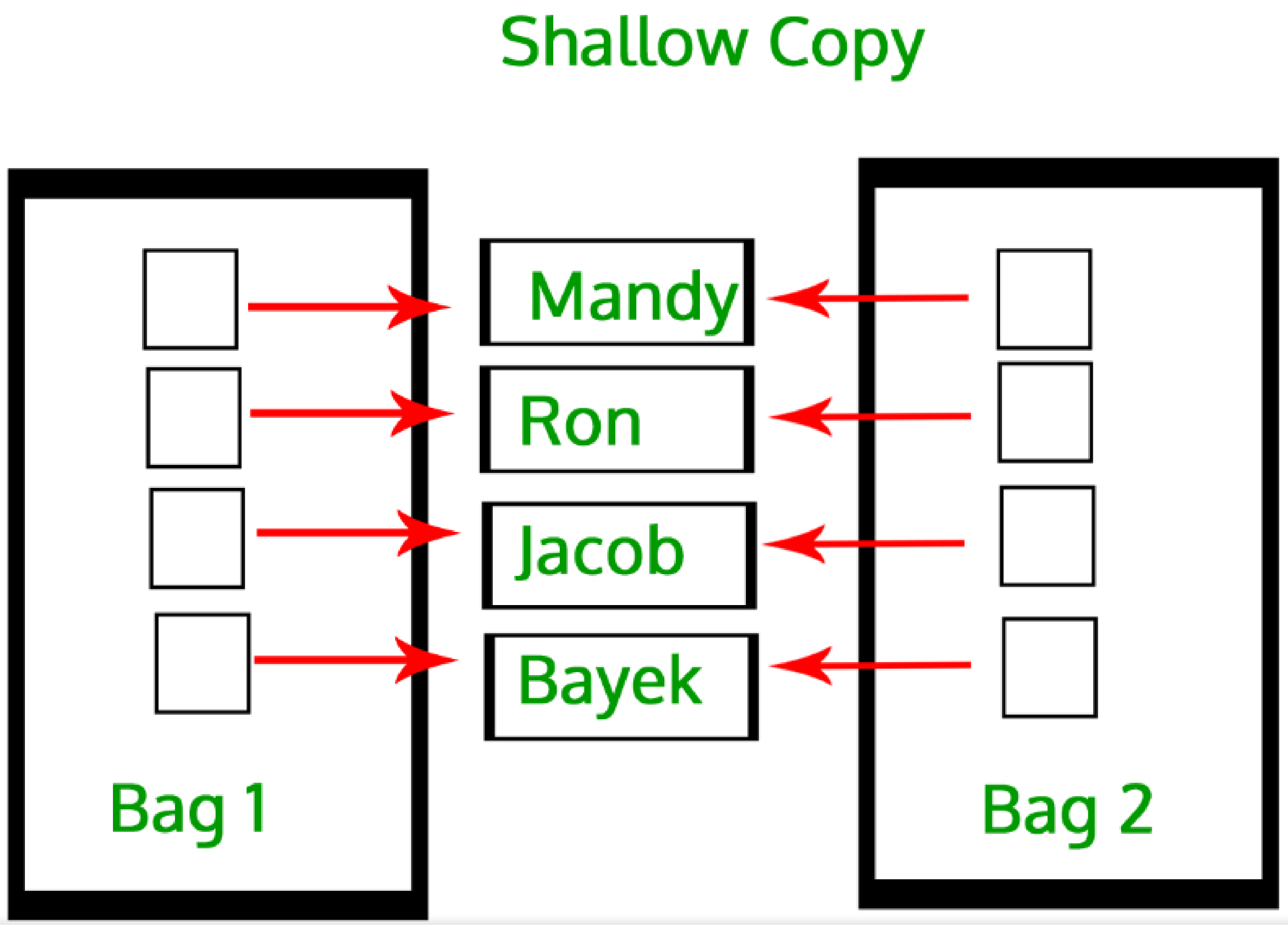


Example:

In the above example, the change made in the list **did not affect** other lists, indicating the list is deeply copied.

What is Shallow copy in Python?

A shallow copy creates a new compound object and then references the objects contained in the original within it, which means it constructs a new collection object and then populates it with references to the child objects found in the original. The copying process does not recurse and therefore won't create copies of the child objects themselves. In the case of shallow copy, a reference of an object is copied into another object. It means that **any changes** made to a copy of an object **do reflect** in the original object. In python, this is implemented using the "**copy()**" function.


Shallow Copy
Mandy
Ron
Jacob
Bayek
Bag 1
Bag 2