**Python Keyword**

Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifiers:

|  |  |
| --- | --- |
| Keyword | Description |
| and | A logical operator |
| as | To create an alias |
| assert | For debugging |
| break | To break out of a loop |
| class | To define a class |
| continue | To continue to the next iteration of a loop |
| def | To define a function |
| del | To delete an object |
| elif | Used in conditional statements, same as else if |
| else | Used in conditional statements |
| except | Used with exceptions, what to do when an exception occurs |
| False | Boolean value, result of comparison operations |
| finally | Used with exceptions, a block of code that will be executed no matter if there is an exception or not |
| for | To create a for loop |
| from | To import specific parts of a module |
| global | To declare a global variable |
| if | To make a conditional statement |
| import | To import a module |
| in | To check if a value is present in a list, tuple, etc. |
| is | To test if two variables are equal |
| lambda | To create an anonymous function |
| None | Represents a null value |
| nonlocal | To declare a non-local variable |
| not | A logical operator |
| or | A logical operator |
| pass | A null statement, a statement that will do nothing |
| raise | To raise an exception |
| return | To exit a function and return a value |
| True | Boolean value, result of comparison operations |
| try | To make a try...except statement |
| while | To create a while loop |
| with | Used to simplify exception handling |
| yield | To end a function, returns a generator |

**Operators:**

operations are actions or calculations that can be performed on values or variables. Python supports a wide range of operators that allow you to perform various operations, such as arithmetic, comparison, logical, assignment, and more. Here are some common types of operations in Python:

### Arithmetic Operations:

1. **Addition:**

result = 5 + 3 # Result is 8

1. **Subtraction:**

result = 7 - 2 # Result is 5

1. **Multiplication:**

result = 4 \* 6 # Result is 24

1. **Division:**

result = 10 / 2 # Result is 5.0 (float)

1. **Floor Division:**

result = 10 // 3 # Result is 3 (integer)

1. **Modulus (Remainder):**

result = 10 % 3 # Result is 1

1. **Exponentiation:**

result = 2 \*\* 3 # Result is 8

### Comparison Operations:

1. **Equal to:**

result = (5 == 5) # Result is True

1. **Not Equal to:**

result = (7 != 5) # Result is True

1. **Greater Than:**

result = (8 > 3) # Result is True

1. **Less Than:**

result = (4 < 6) # Result is True

1. **Greater Than or Equal to:**

result = (10 >= 8) # Result is True

1. **Less Than or Equal to:**

result = (5 <= 3) # Result is False

### Logical Operations:

1. **Logical AND:**

result = (True and False) # Result is False

1. **Logical OR:**

result = (True or False) # Result is True

1. **Logical NOT:**

result = not True # Result is False

### Assignment Operations:

1. **Assignment:**

x = 10

1. **Addition Assignment:**

x += 5 # Equivalent to x = x + 5

1. **Subtraction Assignment:**

y -= 3 # Equivalent to y = y - 3

1. **Multiplication Assignment:**

z \*= 2 # Equivalent to z = z \* 2

1. **Division Assignment:**

w /= 4 # Equivalent to w = w / 4

**Constant**

while the language doesn't have a built-in mechanism to declare constants explicitly, it's a common convention to use uppercase letters for variable names that are intended to be treated as constants. This convention signals to other programmers that the variable should not be modified.

PI = 3.14159

GRAVITY = 9.8

MAX\_CONNECTIONS = 100

# Using constants in code

radius = 5

area = PI \* (radius \*\* 2)

**Strings Operations:**

Strings in Python are sequences of characters and support various operations for manipulation and analysis. Here are some basic operations you can perform on strings:

### 1. Concatenation:

You can concatenate (combine) two or more strings using the + operator:

str1 = "Hello"

str2 = " World"

result = str1 + str2

print(result) # Output: Hello World

### 2. Repetition:

You can repeat a string multiple times using the \* operator:

str1 = "Hello "

result = str1 \* 3

print(result) # Output: Hello Hello Hello

### 3. Accessing Characters:

You can access individual characters in a string using indexing:

my\_string = "Python"

print(my\_string[0]) # Output: P

### 4. Slicing:

You can extract a portion (substring) of a string using slicing:

my\_string = "Python"

substring = my\_string[2:5]

print(substring) # Output: tho

### 5. Length:

You can find the length of a string using the len() function:

my\_string = "Python"

length = len(my\_string)

print(length) # Output: 6

### 6. Conversion:

You can convert a string to uppercase or lowercase:

my\_string = "Python"

uppercase = my\_string.upper()

lowercase = my\_string.lower()

print(uppercase) # Output: PYTHON

print(lowercase) # Output: python

**String Functions:**

Python provides a variety of built-in string functions to perform common operations on strings. Here are some frequently used string functions:

### 1. ****len()****

* Returns the length (number of characters) of a string.

my\_string = "Hello, World!"

length = len(my\_string)

print(length) # Output: 13

### 2. ****capitalize()****

* Converts the first character of a string to uppercase.

my\_string = "hello"

capitalized\_string = my\_string.capitalize()

print(capitalized\_string) # Output: Hello

### 3. ****upper() and lower()****

* upper(): Converts all characters in a string to uppercase.
* lower(): Converts all characters in a string to lowercase.

my\_string = "Hello, World!"

upper\_case = my\_string.upper()

lower\_case = my\_string.lower()

print(upper\_case) # Output: HELLO, WORLD!

print(lower\_case) # Output: hello, world!

### 4. ****count()****

* Returns the number of occurrences of a substring in the string.

my\_string = "apple, orange, banana, orange, grape"

count = my\_string.count("orange")

print(count) # Output: 2

### 5. ****find() and index()****

* find(): Returns the index of the first occurrence of a substring (or -1 if not found).
* index(): Returns the index of the first occurrence of a substring (raises a ValueError if not found).

my\_string = "Hello, World!"

index1 = my\_string.find("World")

index2 = my\_string.index("World")

print(index1) # Output: 7

print(index2) # Output: 7

### 6. ****replace()****

* Replaces occurrences of a substring with another substring.

my\_string = "Hello, World!"

new\_string = my\_string.replace("World", "Universe")

print(new\_string) # Output: Hello, Universe!

### 7. ****strip(), lstrip(), and rstrip()****

* strip(): Removes leading and trailing whitespace.
* lstrip(): Removes leading whitespace.
* rstrip(): Removes trailing whitespace.

my\_string = " Hello, World! "

stripped\_string = my\_string.strip()

print(stripped\_string) # Output: Hello, World!

### 8. ****startswith() and endswith()****

* startswith(): Checks if a string starts with a specified prefix.
* endswith(): Checks if a string ends with a specified suffix.

my\_string = "Hello, World!"

starts\_with\_hello = my\_string.startswith("Hello")

ends\_with\_world = my\_string.endswith("World")

print(starts\_with\_hello) # Output: True

print(ends\_with\_world) # Output: True

### 9. ****split()****

* Splits a string into a list of substrings based on a specified delimiter.

my\_string = "apple, orange, banana"

split\_result = my\_string.split(", ")

print(split\_result) # Output: ['apple', 'orange', 'banana']

### 10. ****join()****

* Joins elements of an iterable into a string using a specified separator.

fruits = ['apple', 'orange', 'banana']

joined\_string = '-'.join(fruits)

print(joined\_string)

**Constant**

A constant is a special type of variable whose value cannot be changed.

# Define constants (using uppercase names)

PI = 3.14159

GRAVITY = 9.8

# Use constants in calculations

radius = 5

area\_of\_circle = PI \* radius \*\* 2

# Display the result

print(f"The area of a circle with radius {radius} is: {area\_of\_circle}")