1. What are keywords in python? Using the keyword library, print all the python keywords.

Answer

In Python, you can use the keyword library to access a list of all the keywords. Here's an example of how to print all the Python keywords using the keyword library:

```
import keyword
all_keywords = keyword.kwlist
 for keyword in all_keywords:
   print(keyword)
```

When you run this code, it will print all the Python keywords in the console:

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

These are the 35 keywords in Python as of my knowledge cutoff in September 2021. Please note that the Python language may introduce new keywords in future versions, so the list may be subject to change.

2. What are the rules to create variables in python?

Answer

- Variable names can contain letters (a-z, A-Z), digits (0-9), and underscores (_).
- They must start with a letter or an underscore.
- Variable names are case-sensitive, meaning myVar and myvar are considered different variables.
- Variable names cannot be the same as Python keywords or reserved words like if, else, for, etc.
- **Valid variable names:** my_variable, count, _name, age1, x.
- **Invalid variable names**: 2name (starts with a digit), my-variable (contains hyphen), for (reserved keyword).
- Python convention recommends using lowercase letters for variable names.
- If the name consists of multiple words, it is common to use underscores to separate them (snake_case).
- **Example**: my_variable, total_count, first_name.
- To assign a value to a variable, use the equals (=) operator.
- **Example**: x = 10, name = "John".
- Variables in Python are dynamically typed, meaning you don't need to explicitly declare their types.
- The type of a variable is determined by the value assigned to it.
- **Example**: x = 10 assigns an integer value to x, while name = "John" assigns a string value.

3. What are the standards and conventions followed for the nomenclature of variables in python to improve code readability and maintainability?

Answer

Use descriptive names: Choose variable names that are meaningful and describe their purpose or contents. Avoid single-letter or cryptic names that may make the code harder to understand.

Use lowercase letters: Variable names are typically written in all lowercase letters. For example: count, name, age.

Separate words with underscores: When a variable name consists of multiple words, it is recommended to use underscores (_) to separate the words. This convention is known as snake_case. For example: total_count, first_name, is_valid.

Avoid using reserved keywords: Do not use Python keywords or reserved words as variable names. For example, do not name a variable for, if, while, etc.

4. What will happen if a keyword is used as a variable name?

Answer

When you attempt to use a keyword as a variable name, Python's interpreter will raise a **SyntaxError** and indicate that the keyword is an invalid identifier.

Example: if = 10

Running this code will result in the following error message:

```
File "<ipython-input-1-1c957e1c40a2>", line 1
if = 10 SyntaxError: invalid syntax
```

5. For what purpose def keyword is used?

Answer

In Python, the def keyword is used to define a function.

When you use the def keyword followed by a function name, you are declaring a new function and defining its behavior.

6. What is the operation of this special character '\'?

Answer

In Python, the special character \ is called the backslash or escape character.

The backslash is used to create escape sequences, which represent special characters within string literals.

For example:

\': Represents a single quote character.
\": Represents a double quote character.
\\: Represents a backslash character.
\n: Represents a newline character.
\t: Represents a tab character.
\r: Represents a carriage return character.
\b: Represents a backspace character.

7. Give an example of the following conditions:
(i) Homogeneous list
(ii) Heterogeneous set
(iii) Homogeneous tuple

Answer

(i) Homogeneous list:

A homogeneous list in Python is a list that contains elements of the same data type. Here's an example of a homogeneous list containing integers:

numbers = [1, 2, 3, 4, 5]

In this example, the list numbers contains only integer values, making it a homogeneous list.

(ii) Heterogeneous set:

A heterogeneous set in Python is a set that can contain elements of different data types. Here's an example of a heterogeneous set:

data = {1, 'apple', 3.14, True}

In this example, the set data contains elements of various data types: an integer (1), a string ('apple'), a floating-point number (3.14), and a boolean (True).

(iii) Homogeneous tuple:

A homogeneous tuple in Python is a tuple that contains elements of the same data type. Here's an example of a homogeneous tuple containing strings:

names = ('Alice', 'Bob', 'Charlie', 'Dave')

In this example, the tuple names contains only string values, making it a homogeneous tuple.

8. Explain the mutable and immutable data types with proper explanation & examples.

Answer

Immutable Data Types:

Immutable data types are those whose values cannot be modified once they are assigned. If you want to modify an immutable object, you need to create a new object with the desired changes.

Examples of immutable data types in Python include:

Numeric types: int, float, complex
String: str
Tuple: tuple
FrozenSet: frozenset

Here's an example to illustrate the immutability of the string data type:

```
name = "Alice"
print(name)  # Output: Alice
# Trying to modify the string
name[0] = 'B'  # Raises a TypeError: 'str' object does not support item assignment
```

In this example, the variable name is assigned the string "Alice". When we attempt to modify the first character of the string, we get a TypeError because strings are immutable. To change the string, we need to create a new string object.

Mutable Data Types:

Mutable data types, on the other hand, allow modifications to their values after they are created. You can change, add, or remove elements within a mutable object without creating a new object.

Examples of mutable data types in Python include:

List: list
Dictionary: dict
Set: set

Here's an example to demonstrate the mutability of a list:

```
numbers = [1, 2, 3, 4, 5]
print(numbers)  # Output: [1, 2, 3, 4, 5]
# Modifying the list
numbers[0] = 10
numbers.append(6)
print(numbers)  # Output: [10, 2, 3, 4, 5, 6]
```

In this example, the list numbers is initially assigned with the values [1, 2, 3, 4, 5]. We can modify the first element of the list and add a new element using the = assignment and append() method, respectively. The changes are reflected in the original list itself.

9. Write a code to create the given structure using only for loop.

```
    *
   ***
  *****
 *******
*********
```

Answer

```
rows = 5
spaces = rows - 1
stars = 1
for i in range(rows):
    for j in range(spaces):
        print(" ", end="")
    for j in range(stars):
        print("*", end="")
    print()
    spaces -= 1
    stars += 2
```

10. Write a code to create the given structure using while loop.

```
|||||||||
 |||||||
  |||||
   |||
    |
```

Answer

```
rows = 5
spaces = 0
stars = rows * 2 - 1
while rows > 0:
    i = 0
    while i < spaces:
        print(" ", end="")
        i += 1
    i = 0
    while i < stars:
        print("|", end="")
        i += 1
    print()
    spaces += 1
    stars -= 2
    rows -= 1
```