Keyword List

Here's a list of Python keywords or reserved words along with a brief summary and an example for each:

1. **and**: Logical operator for conjunction. Returns True if both operands are true.

   ```python
   x = True
   y = False
   result = x and y  # result is False
   ```

2. **as**: Used to create an alias when importing a module or a specific attribute.

   ```python
   import math as m
   print(m.sqrt(25))  # Output: 5.0
   ```

3. **assert**: Used for debugging purposes. Raises an error if the given expression is False.

   ```python
   x = 5
   assert x > 0, "Value must be greater than 0"
   ```

4. **break**: Terminates the nearest enclosing loop or `if` statement.

   ```python
   for i in range(5):
       if i == 3:
           break
       print(i)
   # Output: 0, 1, 2
   ```

5. **class**: Declares a new class, a blueprint for creating objects.

   ```python
   class Dog:
       def __init__(self, name):
           self.name = name
   ```

6. **continue**: Jumps to the next iteration of the nearest enclosing loop.

   ```python
   ```

```
for i in range(5):
    if i == 2:
        continue
    print(i)
# Output: 0, 1, 3, 4
```

7. **def**: Defines a function.

```python
def greet(name):
    print("Hello, " + name + "!")
```

8. **del**: Deletes a variable or an element from a collection (such as a list).

```python
my_list = [1, 2, 3]
del my_list[1]
print(my_list)  # Output: [1, 3]
```

9. **elif**: Stands for "else if" and is used in conditional statements.

```python
x = 10
if x > 10:
    print("x is greater than 10")
elif x == 10:
    print("x is equal to 10")
else:
    print("x is less than 10")
```

10. **else**: Specifies a block of code to be executed if the conditional expression in an `if` statement is False.

```python
x = 5
if x > 10:
    print("x is greater than 10")
else:
    print("x is not greater than 10")
```

11. **except**: Catches exceptions in a `try` block.

```python
try:
```

```
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

12. **False**: Boolean value representing false.

```python
x = False
```

13. **finally**: Specifies a block of code to be executed regardless of whether an exception is raised or not.

```python
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
finally:
    print("This will always execute.")
```

14. **for**: Used for looping over a sequence (such as a list or string).

```python
for i in range(5):
    print(i)
# Output: 0, 1, 2, 3, 4
```

15. **from**: Used to import specific attributes or functions from a module.

```python
from math import sqrt
print(sqrt(25))  # Output: 5.0
```

16. **global**: Declares a global variable inside a function.

```python
def set_global():
    global x
    x = 10

set_global()
print(x)  # Output: 10
```

17. **if**: Conditionally executes a block of code.

```python
x = 5
if x > 0:
    print("x is positive")
```

18. **import**: Imports a module.

```python
import math
print(math.sqrt(25))  # Output: 5.0
```

19. **in**: Checks if a value exists in a sequence.

```python
my_list = [1, 2, 3]
if 2 in my_list:
    print("2 is in the list")
```

20. **is**: Tests object identity.

```python
x = [1, 2, 3]
y = [1, 2, 3]
if x is y:
    print("x and y refer to the same object")
else:
    print("x and y are different objects")
```

21. **lambda**: Creates an anonymous function.

```python
add = lambda x, y: x + y
print(add(2, 3))  # Output: 5
```

22. **None**: Represents the absence of a value or a null value.

```python
x = None
```

23. **nonlocal**: Declares a variable to be nonlocal, meaning it is not defined in the current function but in an enclosing function.

```python
def outer_function():
    x = 10
    def inner_function():
        nonlocal x
        x += 5
    inner_function()
    print(x)  # Output: 15
```

24. **not**: Logical operator for negation.

```python
x = True
result = not x  # result is False
```

25. **or**: Logical operator for disjunction. Returns True if at least one operand is true.

```python
x = True
y = False
result = x or y  # result is True
```

26. **pass**: A null operation. Used as a placeholder where syntactically some code is required but no action is desired or necessary.

```python
def my_function():
    pass
```

27. **raise**: Raises an exception.

```python
raise ValueError("This is a custom error message")
```

28. **return**: Exits a function and returns a value.

```python
def add(x, y):
    return x + y

result = add(3, 5)
print(result)  # Output: 8
```

29. **True**: Boolean value representing true.

    ```python
    x = True
    ```

30. **try**: Specifies a block of code to be tested for errors.

    ```python
    try:
        result = 10 / 0
    except ZeroDivisionError:
        print("Cannot divide by zero!")
    ```

31. **while**: Creates a loop that continues until a certain condition is False.

    ```python
    i = 0
    while i < 5:
        print(i)
        i +=

1
    # Output: 0, 1, 2, 3, 4
    ```

32. **with**: Simplifies resource management (like file handling) using a context manager.

    ```python
    with open("example.txt", "r") as file:
        content = file.read()
    ```

33. **yield**: Pauses a function, saving its state to be resumed later.

    ```python
    def my_generator():
        yield 1
        yield 2
        yield 3

    gen = my_generator()
    print(next(gen))  # Output: 1
    ```