1. Break in a loop

Stops the loop when a condition is met.

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

Output:

```
0
1
2
3
4
```

2. Continue in a loop

Skips the current iteration and continues to the next iteration when a condition is met.

```
for i in range(10):
    if i == 5:
        continue
    print(i)
```

Output:

```
0
1
2
3
4
6
7
8
9
```

3. Pass in a loop

Does nothing when a condition is met (a placeholder).

```
for i in range(10):
    if i == 5:
        pass
    print(i)
```

Output:

```
0
1
2
3
4
5
6
7
8
9
```

4. Function without arguments

Defines a function without parameters and prints a message.

```
def greet():
    print("Hello, World!")

greet()
```

Output:

```
Hello, World!
```

5. Function with arguments

Defines a function that accepts parameters and returns a value.

```
def add(a, b):
    return a + b

print(add(2, 3))
```

Output:

```
5
```

6. Break inside a nested loop

Stops both loops when a condition is met inside the inner loop.

```
for i in range(5):
    for j in range(5):
        if j == 3:
            break
        print(i, j)
```

Output:

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
3 0
3 1
3 2
4 0
4 1
```

```
4 2
```

7. Return statement in a function

Defines a function that returns a value.

```
def square(x):
    return x * x

print(square(4))
```

Output:

```
16
```

8. Multiple conditions with break

Demonstrates breaking out of a loop with multiple conditions.

```
for i in range(10):
    if i == 3:
        break
    elif i == 7:
        break
    print(i)
```

Output:

```
0
1
2
```

9. Using pass in function

Uses `pass` to create a placeholder in a function.

```
def placeholder():
    pass
```

```
print("Function with pass.")
```

Output:

```
Function with pass.
```

10. Recursion in a function

Demonstrates recursion to calculate the factorial of a number.

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)

print(factorial(5))
```

Output:

```
120
```

1. Break in a loop with an else clause

Using `else` with a loop when it's not interrupted by `break`.

```
for i in range(5):
    if i == 3:
        break
    print(i)
else:
    print("Loop completed without break.")
```

Output:

```
0
1
2
```

2. Continue with an else clause

Using `else` after a loop to print a message when no `continue` occurs.

```
for i in range(5):
    if i == 3:
        continue
    print(i)
else:
    print("Loop finished without continue.")
```

Output:

```
0
1
2
4
Loop finished without continue.
```

3. Pass used in a function to skip code

A function that uses `pass` as a placeholder for unimplemented code.

```
def my_function():
    pass

print("Function with pass executed.")
```

Output:

```
Function with pass executed.
```

4. Recursive function to calculate Fibonacci sequence

A recursive function to find the nth Fibonacci number.

```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

print(fibonacci(6))
```

Output:

```
8
```

5. Break with a while loop

Using `break` to stop a while loop when a condition is met.

```
i = 0
while i < 10:
    if i == 6:
        break
    print(i)
    i += 1
```

Output:

```
0
1
2
3
4
5
```

6. Continue with a while loop

Using `continue` to skip an iteration in a while loop.

```
i = 0
while i < 10:
    i += 1
    if i == 5:
        continue
    print(i)
```

Output:

```
1
2
3
4
6
7
8
9
10
```

7. Function with multiple return statements

A function that checks whether a number is even or odd and returns accordingly.

```
def check_number(n):
```

```
    if n % 2 == 0:
        return "Even"
    else:
        return "Odd"

print(check_number(7))
print(check_number(10))
```

Output:

```
Odd
Even
```

8. Break out of a nested loop with a flag

Using a flag to break out of nested loops.

```
found = False
for i in range(5):
    for j in range(5):
        if i == 3 and j == 2:
            found = True
            break
    if found:
        break
print("Found at position (3, 2)")
```

Output:

```
Found at position (3, 2)
```

9. Pass in a class method

Using `pass` in a class method as a placeholder for future code.

```
class MyClass:
    def method(self):
        pass
```

```
obj = MyClass()
obj.method()
print("Method with pass called.")
```

Output:

```
Method with pass called.
```

10. Function with variable-length arguments

A function that accepts a variable number of arguments and calculates the sum.

```
def sum_numbers(*args):
    return sum(args)

print(sum_numbers(1, 2, 3, 4))
print(sum_numbers(5, 6, 7))
```

Output:

```
10
18
```