


```

from random import randint
def generate_random_lengths():
    """
    generate random lengths
    """
    # generate random lengths
    a = randint(1, 10)
    b = randint(1, 10)
    c = randint(1, 10)
    return a, b, c
def is_valid_triangle(a, b, c):
    """
    This function checks the most fundamental of a triangle, given the lengths
    of its three sides.
    """
    # check if a, b, c are positive
    if a <= 0 or b <= 0 or c <= 0:
        return "Invalid Triangle"
    # check if a, b, c satisfy the triangle inequality
    if a + b <= c or a + c <= b or b + c <= a:
        return "Invalid Triangle"
    return "Valid Triangle"
def main():
    """
    Main function to generate random lengths and check if they form a valid triangle.
    """
    # generate random lengths
    a, b, c = generate_random_lengths()
    # check if they form a valid triangle
    result = is_valid_triangle(a, b, c)
    print(f"a = {a}, b = {b}, c = {c}")
    print(f"Result: {result}")
if __name__ == "__main__":
    main()

```

```

"""
This module defines the most fundamental of a triangle, given the lengths
of its three sides.
"""
def is_valid_triangle(a, b, c):
    """
    This function checks the most fundamental of a triangle, given the lengths
    of its three sides.
    """
    # check if a, b, c are positive
    if a <= 0 or b <= 0 or c <= 0:
        return "Invalid Triangle"
    # check if a, b, c satisfy the triangle inequality
    if a + b <= c or a + c <= b or b + c <= a:
        return "Invalid Triangle"
    return "Valid Triangle"

```

```

"""
This module defines the most fundamental of a triangle, given the lengths
of its three sides.
"""

```

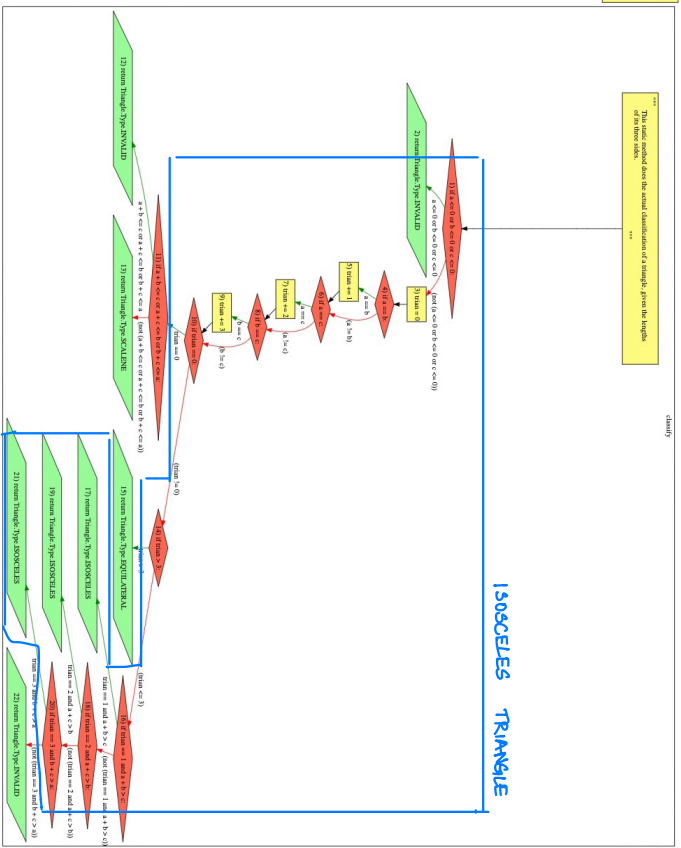
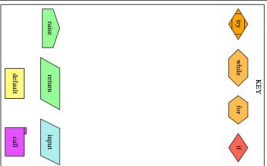
is_valid_triangle

is_valid_triangle

```

"""
This module defines the most fundamental of a triangle, given the lengths
of its three sides.
"""

```



is_valid_triangle

An implementation that classifies triangles

Statistical method

```
return TriangleType.INV;
return 0;
```

$$\begin{aligned} \text{sum} &= 0 \\ \text{for } i &= 1 \end{aligned}$$
if $a = c$:
$$\zeta = +\infty$$

```

if a + b < 0
    return 1
endif

```

```
return TriangleType.SCALES;
if (ratio > 1) {
```

if $\text{trial} = 1$ and $a + b > c$;
 $b < a + b \text{ and } 1 = \text{trial}$;

```
return TriagleType.ISOSCELES
```

```
return Image::Type::ISOSURF;
return Image::Type::INVALID;
```

INVALID = 0

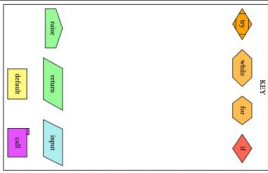
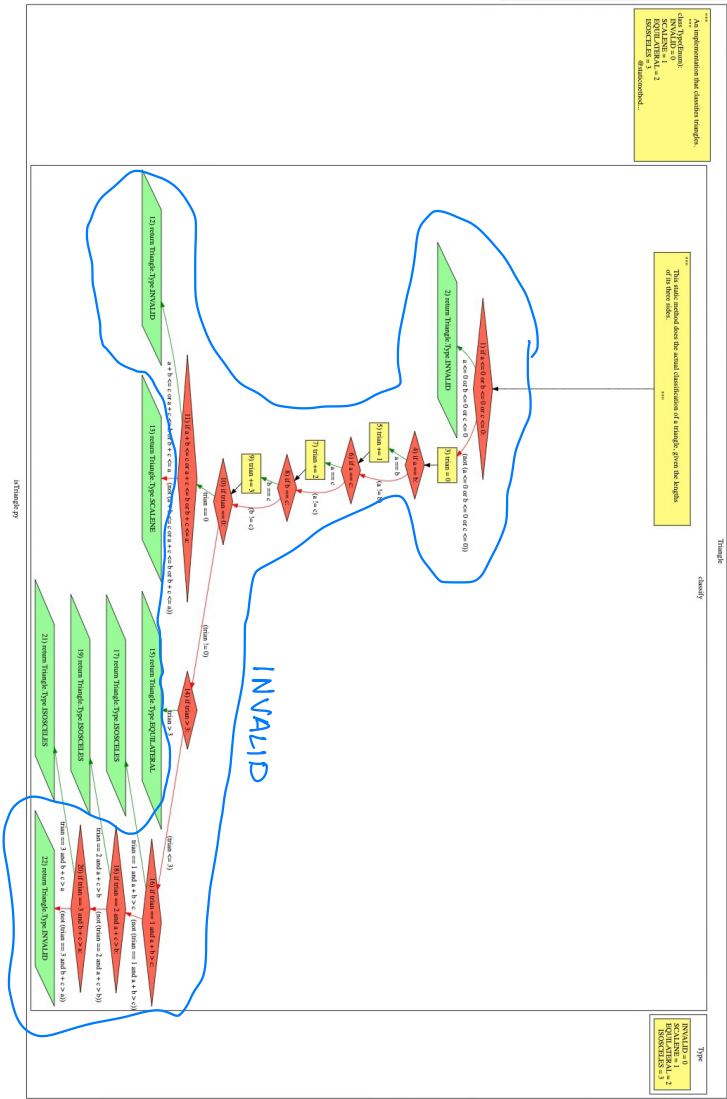
EQUILATERAL = 2
ISOSCELES = 1

This static method does the actual classification of a triangle, given the lengths of its three sides.

classif.

 $\text{d}(L)$

SCALED = 1
EQUILATERAL = 2
ISOSCELES = 3



KEY

- if
- for
- while
- try
- raise
- return
- import
- call
- default

