```
PY_SIMPLE_COMPLETION_INSTRUCTION = (
    "# Write the body of this function only."
)
PY_REFLEXION_COMPLETION_INSTRUCTION = (
    "You are a Python writing assistant. You will be given your past"
    " function implementation, a series of unit tests, and a hint to"
    " change the implementation appropriately. Write your full"
    " implementation (restate the function signature).\n\n-----"
)
PY_SELF_REFLECTION_COMPLETION_INSTRUCTION = (
    "You are a Python writing assistant. You will be given a function"
    " implementation and a series of unit tests. Your goal is to"
    " write a few sentences to explain why your implementation is"
    " wrong as indicated by the tests. You will need this as a hint"
    " when you try again later. Only provide the few sentence"
    " description in your answer, not the implementation.\n\n-----"
)
USE_PYTHON_CODEBLOCK_INSTRUCTION = (
    "Use a Python code block to write your response. For"
    " example:\n```python\nprint('Hello world!')\n```"
)

PY_SIMPLE_CHAT_INSTRUCTION = (
    "You are an AI that only responds with python code, NOT ENGLISH."
    " You will be given a function signature and its docstring by the"
    " user. Write your full implementation (restate the function"
```

```
    " signature)."
)
PY_SIMPLE_CHAT_INSTRUCTION_V2 = (
    "You are an AI that only responds with only python code. You will"
    " be given a function signature and its docstring by the user."
    " Write your full implementation (restate the function"
    " signature)."
)
PY_REFLEXION_CHAT_INSTRUCTION = (
    "You are an AI Python assistant. You will be given your past"
    " function implementation, a series of unit tests, and a hint to"
    " change the implementation appropriately. Write your full"
    " implementation (restate the function signature)."
)
PY_REFLEXION_CHAT_INSTRUCTION_V2 = (
    "You are an AI Python assistant. You will be given your previous"
    " implementation of a function, a series of unit tests results,"
    " and your self-reflection on your previous implementation. Write"
    " your full implementation (restate the function signature)."
)
PY_REFLEXION_FEW_SHOT_ADD = '''Example 1:
[previous impl]:
```python
def add(a: int, b: int) -> int:
    """
    Given integers a and b, return the total value of a and b.
```
```

```
    """
    return a - b
```

[unit test results from previous impl]:

Tested passed:

Tests failed:

assert add(1, 2) == 3 # output: -1

assert add(1, 2) == 4 # output: -1

[reflection on previous impl]:

The implementation failed the test cases where the input integers are 1 and 2. The issue arises because the code does not add the two integers together, but instead subtracts the second integer from the first. To fix this issue, we should change the operator from `-` to `+` in the return statement. This will ensure that the function returns the correct output for the given input.

[improved impl]:

```python
def add(a: int, b: int) -> int:
    """
    Given integers a and b, return the total value of a and b.
    """
    return a + b
```

'''

PY_REFLEXION_FEW_SHOT = '''Example 1:

[previous impl]:

```python

from typing import *

def fullJustify(words: List[str], maxWidth: int) -> List[str]:

    """

    Given an array of words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.

    You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces `' '` when necessary so that each line has exactly maxWidth characters.

    Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

    For the last line of text, it should be left justified and no extra space is inserted between words.

    Note:

    A word is defined as a character sequence consisting of non-space characters only.

    Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.

    The input array `words` contains at least one word.

    """

    res = []

    cur_line = []

    cur_len = 0


    for word in words:

        if cur_len + len(word) + len(cur_line) > maxWidth:
```

```
            if len(cur_line) == 1:
                res.append(cur_line[0] + ' ' * (maxWidth - cur_len))
            else:
                spaces = maxWidth - cur_len
                space_between = spaces // (len(cur_line) - 1)
                extra_spaces = spaces % (len(cur_line) - 1)
                line = ''
                for i, w in enumerate(cur_line[:-1]):
                    line += w + ' ' * (space_between + (i < extra_spaces))
                line += cur_line[-1]
                res.append(line)
            cur_line = []
            cur_len = 0
        cur_line.append(word)
        cur_len += len(word)

    last_line = ' '.join(cur_line)
    last_line += ' ' * (maxWidth - len(last_line))
    res.append(last_line)

    return res
```

[unit test results from previous impl]:

Tested passed:

Tests failed:

assert fullJustify([], 10) == [] # output: ['   ']

assert fullJustify([], 0) == [] # output: ['']

[reflection on previous impl]:

The implementation failed the test cases where the input list of words is empty. The issue arises because the code does not handle the case where there are no words to process. As a result, it still appends a line with spaces to the result list, even when there are no words. To fix this issue, we should add a condition at the beginning of the function to check if the input list is empty, and return an empty list if it is. This will ensure that the function returns the correct output for empty input lists.

[improved impl]:

```python
from typing import *
def fullJustify(words: List[str], maxWidth: int) -> List[str]:
    """
    Given an array of words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.
    You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces `' '` when necessary so that each line has exactly maxWidth characters.
    Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.
    For the last line of text, it should be left justified and no extra space is inserted between words.
    Note:
    A word is defined as a character sequence consisting of non-space characters only.
```

```
Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.

The input array `words` contains at least one word.
"""

if not words:
    return []


res = []
cur_line = []
cur_len = 0


for word in words:
    if cur_len + len(word) + len(cur_line) > maxWidth:
        if len(cur_line) == 1:
            res.append(cur_line[0] + ' ' * (maxWidth - cur_len))
        else:
            spaces = maxWidth - cur_len
            space_between = spaces // (len(cur_line) - 1)
            extra_spaces = spaces % (len(cur_line) - 1)
            line = ''
            for i, w in enumerate(cur_line[:-1]):
                line += w + ' ' * (space_between + (i < extra_spaces))
            line += cur_line[-1]
            res.append(line)
        cur_line = []
        cur_len = 0
    cur_line.append(word)
```

```
        cur_len += len(word)

    last_line = ' '.join(cur_line)
    last_line += ' ' * (maxWidth - len(last_line))
    res.append(last_line)

    return res
```

END EXAMPLES

'''

PY_SELF_REFLECTION_CHAT_INSTRUCTION = (
    "You are a Python programming assistant. You will be given a"
    " function implementation and a series of unit tests. Your goal"
    " is to write a few sentences to explain why your implementation"
    " is wrong as indicated by the tests. You will need this as a"
    " hint when you try again later. Only provide the few sentence"
    " description in your answer, not the implementation."
)

PY_SELF_REFLECTION_CHAT_INSTRUCTION_V2 = (
    "You are a Python programming assistant. You will be given a"
    " function implementation and a series of unit test results. Your"
    " goal is to write a few sentences to explain why your"
    " implementation is wrong as indicated by the tests. You will"
    " need this as guidance when you try again later. Only provide"
    " the few sentence description in your answer, not the"

```
    " implementation. You will be given a few examples by the user."
)
PY_SELF_REFLECTION_FEW_SHOT = """Example 1:
[function impl]:
```python
def longest_subarray_with_sum_limit(nums: List[int], target: int) -> List[int]:
    n = len(nums)
    left, right = 0, 0
    max_length = 0
    current_sum = 0
    result = []
    while right < n:
        current_sum += nums[right]
        while current_sum > target:
            current_sum -= nums[left]
            left += 1
        if right - left + 1 >= max_length:
            max_length = right - left + 1
            result = nums[left:right+1]
        right += 1
    return result
```
[unit test results]:
Tests passing:
assert longest_subarray_with_sum_limit([1, 2, 3, 4, 5], 8) == [1, 2, 3]
assert longest_subarray_with_sum_limit([1, 2, 3, 4, 5], 15) == [1, 2, 3, 4, 5]
```

assert longest_subarray_with_sum_limit([1, -1, 2, -2, 3, -3], 2) == [1, -1, 2, -2, 3]

assert longest_subarray_with_sum_limit([], 10) == []

assert longest_subarray_with_sum_limit([], 0) == []

assert longest_subarray_with_sum_limit([], -5) == []

Tests failing:

assert longest_subarray_with_sum_limit([5, 6, 7, 8, 9], 4) == [] # output: [5]

[self-reflection]:

The implementation failed the where no subarray fulfills the condition. The issue in the implementation is due to the use of >= instead of > in the condition to update the result. Because of this, it returns a subarray even when the sum is greater than the target, as it still updates the result when the current subarray length is equal to the previous longest subarray length. To overcome this error, we should change the condition to only update the result when the current subarray length is strictly greater than the previous longest subarray length. This can be done by replacing >= with > in the condition.

Example 2:

[function impl]:

```python
def longest_subarray_with_sum_limit(nums: List[int], target: int) -> List[int]:
    n = len(nums)
    left, right = 0, 0
    max_length = 0
    current_sum = 0
    result = []
    while current_sum + nums[right] <= target:
        current_sum += nums[right]
```

```
        right += 1
    while right < n:
        current_sum += nums[right]
        while current_sum > target:
            current_sum -= nums[left]
            left += 1
        if right - left + 1 > max_length:
            max_length = right - left + 1
            result = nums[left:right+1]
        right += 1
    return result
```

[unit test results]:

Tests passing:

assert longest_subarray_with_sum_limit([], 10) == []

assert longest_subarray_with_sum_limit([], 0) == []

assert longest_subarray_with_sum_limit([], -5) == []

Tests failing:

assert longest_subarray_with_sum_limit([1, 2, 3, 4, 5], 8) == [1, 2, 3] # output: list index out of range

assert longest_subarray_with_sum_limit([1, 2, 3, 4, 5], 15) == [1, 2, 3, 4, 5] # output: list index out of range

assert longest_subarray_with_sum_limit([5, 6, 7, 8, 9], 4) == [] # output: list index out of range

assert longest_subarray_with_sum_limit([1, -1, 2, -2, 3, -3], 2) == [1, -1, 2, -2, 3] # output: list index out of range

[self-reflection]:

The implementation failed 4 out of the 7 test cases due to an IndexError. The issue stems from the

while loop while current_sum + nums[right] <= target:, which directly accesses nums[right] without checking if right is within the bounds of the list. This results in a runtime error when right goes beyond the list length. To overcome this error, we need to add a bounds check for the right variable in the mentioned while loop. We can modify the loop condition to while right < len(nums) and current_sum + nums[right] <= target:. This change will ensure that we only access elements within the bounds of the list, thus avoiding the IndexError.

END OF EXAMPLES

"""

PY_TEST_GENERATION_FEW_SHOT = """Examples:

func signature:

```
def add3Numbers(x, y, z):
    \"\"\" Add three numbers together.
    This function takes three numbers as input and returns the sum of the three numbers.
    \"\"\"
```

unit tests:

```
assert add3Numbers(1, 2, 3) == 6
assert add3Numbers(-1, 2, 3) == 4
assert add3Numbers(1, -2, 3) == 2
assert add3Numbers(1, 2, -3) == 0
assert add3Numbers(-3, -2, -1) == -6
assert add3Numbers(0, 0, 0) == 0
```

"""

PY_TEST_GENERATION_COMPLETION_INSTRUCTION = f"""You are an AI coding assistant that can write unique, diverse, and intuitive unit tests for functions given the signature and docstring.

{PY_TEST_GENERATION_FEW_SHOT}"""

PY_TEST_GENERATION_CHAT_INSTRUCTION = """You are an AI coding assistant that can write unique, diverse, and intuitive unit tests for functions given the signature and docstring."""