四舍五入和位数保留

```
result = round(3.14159, 2)  # 结果为 3.14（遵循四舍六入五凑偶）
number = 3.14159
print(f'{number:.2f}')  # 输出结果为 "3.14"
```

进制转化

```
#二进制
decimal_number = 10
binary_string = bin(decimal_number)  # 结果为 '0b1010'
print(binary_string)
#八进制
decimal_number = 10
octal_string = oct(decimal_number)  # 结果为 '0o12'
print(octal_string)
#十六进制
decimal_number = 10
hexadecimal_string = hex(decimal_number)  # 结果为 '0xa'
print(hexadecimal_string)
#转化为十进制
binary_string = '1010'
decimal_number = int(binary_string, 2)  # 结果为 10，基底可以为2（即进制数）
print(decimal_number)
#请注意，所有的转换都要求输入是一个有效的表示形式，并且基数必须在2到36之间（因为这是所有可用数字和字母能表示的最大进制）。如果输入非法或者基数超出范围，将会抛出异常。
```

判断全真和全假

`all()` 函数用于检查可迭代对象的所有元素是否都为真，如果都为真则返回 `True`，否则返回 `False`。

`any()` 函数用于检查可迭代对象中是否有任何一个元素为真，如果至少有一个元素为真，则返回 `True`，否则返回 `False`。

下面是如何使用这两个函数来判断一个列表中的元素是否全为真或全为假：

```
def check_all_elements(iterable):
    # 判断是否全为真
    all_true = all(iterable)
    # 判断是否全为假
    all_false = not any(iterable)

    if all_true:
        return "All elements are true."
    elif all_false:
        return "All elements are false."
    else:
        return "Elements are mixed."

# 示例
elements = [True, True, True]
print(check_all_elements(elements))  # 输出: All elements are true.
```

```
elements = [False, False, False]
print(check_all_elements(elements))  # 输出: All elements are false.

elements = [True, False, True]
print(check_all_elements(elements))  # 输出: Elements are mixed.
```

这个函数首先检查所有元素是否都为真，如果不是，则检查是否有任何元素为真。根据这两个条件的结果，它返回相应的信息。注意这里的“假”是指Python中视为`False`的任何东西，包括但不限于`0`，`None`，`False`，空序列（如`[]`，`{}`，`''`等）。

对数运算

看起来您可能想要使用Python来计算自然对数。在Python中，您可以使用`math`模块里的`log()`函数来计算一个数的自然对数。下面是一个简单的例子：

```
import math

# 计算 e 的自然对数，其中 e 是自然对数的底数，大约等于 2.71828
ln_e = math.log(math.e)
print(ln_e)  # 输出应该是 1.0，因为 log(e, e) = 1

# 计算 10 的自然对数
ln_10 = math.log(10)
print(ln_10)  # 输出结果将接近 2.302585092994046
```

如果您想计算以其他底数的对数，可以这样写：

```
# 计算 16 以 2 为底的对数
log2_16 = math.log(16, 2)
print(log2_16)  # 输出应该是 4.0，因为 2^4 = 16

# 或者使用 log() 的变种 log2() 来计算以 2 为底的对数
log2_16b = math.log2(16)
print(log2_16b)  # 同样输出 4.0
```

质数筛选：

欧拉筛法（Euler Sieve）是一种改进的筛法，相比传统的埃拉托斯特尼筛法，它减少了不必要的操作，从而提高了效率。欧拉筛法的核心思想是利用已知的质数来标记合数，并且每个合数只被标记一次。

以下是使用欧拉筛法在Python中筛选质数的基本步骤：

1. 初始化一个布尔数组`is_prime[]`，表示每个位置上的数是否为质数。
2. 遍历数组，对于每一个当前位置`i`，如果`is_prime[i]`为`True`，则认为`i`是一个质数。
3. 对于每个质数`i`，遍历`i`的倍数，并标记这些倍数为合数。为了避免重复标记，从`i * i`开始标记。
4. 记录每个质数及其倍数的最小质因数（最小的能够整除该数的质数）。

下面是使用欧拉筛法实现的Python代码：

```
def euler_sieve(limit):
    if limit < 2:
```

```
        return []

    # 初始化数组
    is_prime = [True] * (limit + 1)
    min_prime_factor = [0] * (limit + 1)

    # 0 和 1 不是质数
    is_prime[0] = is_prime[1] = False

    # 遍历数组，找出质数并标记它们的倍数
    for i in range(2, limit + 1):
        if is_prime[i]:  # 如果 i 是质数
            min_prime_factor[i] = i
            # 标记 i 的倍数
            for j in range(i * i, limit + 1, i):
                if is_prime[j]:
                    is_prime[j] = False
                    min_prime_factor[j] = i

    # 收集所有质数
    primes = [i for i, prime in enumerate(is_prime) if prime]
    return primes

# 示例：找出小于等于100的所有质数
primes = euler_sieve(100)
print(primes)
```

在这个实现中，我们不仅标记了合数，还记录了每个合数的最小质因数。这在某些情况下可能会有用，比如在分解质因数时。

需要注意的是，虽然欧拉筛法在理论上减少了不必要的操作，但在实际应用中，其效率提升相对于标准的埃拉托斯特尼筛法并不是特别显著。然而，在处理较大范围的数据时，这种减少冗余的操作可以带来一定的性能改善。

双关键词排序

如果你想根据多个键来排序列表，可以在 `key` 参数中提供一个函数，该函数返回一个包含所有排序依据的元组。Python 的排序算法是稳定的，因此它会在第一个排序键相同的情况下保持原有的顺序，或者根据第二个排序键继续排序。

假设你有一个列表，其中每个元素都是一个字典，包含了姓名和年龄，你想首先按年龄排序，然后在年龄相同时再按姓名排序：

```
people = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'David', 'age': 30},
    {'name': 'Carol', 'age': 25}
]

# 按年龄排序，在年龄相同时按姓名排序
sorted_people = sorted(people, key=lambda person: (person['age'],
person['name']))
print(sorted_people)
```

```
# 输出:
# [
#     {'name': 'Bob', 'age': 25},
#     {'name': 'Carol', 'age': 25},
#     {'name': 'Alice', 'age': 30},
#     {'name': 'David', 'age': 30}
# ]
```

在这个例子中，`lambda person: (person['age'], person['name'])` 返回一个元组 `(age, name)`，用于排序。这会让列表首先按照 `age` 进行排序，当 `age` 相同时，则按照 `name` 排序。

如果你想要降序排序，可以在 `reverse=True` 的情况下，也可以对元组中的单个元素进行反转：

```
# 先按年龄降序，再按姓名升序排序
sorted_people_desc = sorted(people, key=lambda person: (-person['age'],
person['name']))
print(sorted_people_desc)

# 或者直接使用 reverse 参数
sorted_people_desc2 = sorted(people, key=lambda person: (person['age'],
person['name']), reverse=True)
print(sorted_people_desc2)
```

这样你就可以根据多个条件来对列表中的元素进行排序了。