复习题

3. 时间复杂度和空间复杂度

时间复杂度：衡量算法执行所需的时间，通常通过分析输入规模 n 与算法操作次数之间的关系来表示。例如：常见的时间复杂度有 O(1)、O(n)、O(n^2) 等，表示随着输入规模增加，算法的执行时间如何变化。

空间复杂度：衡量算法运行时所需的内存空间。同样通过分析输入规模与所需存储空间之间的关系来表示。空间复杂度描述了算法运行过程中所需的额外内存。

4. 算法的作用

算法的主要作用是通过系统化的步骤和逻辑，解决各种计算问题，优化资源使用，并提高程序的效率。

5. 常用的评判算法效率的方法

时间复杂度分析：通过分析算法的操作步骤数量随输入规模增长的情况，评估其效率。

空间复杂度分析：通过分析算法的存储需求随输入规模的变化情况，评估其内存使用情况。

6. 评判算法复杂度的方法

大 O 符号：用于表示算法的最坏情况复杂度。例如，O(n) 表示线性时间复杂度。

小 o 符号：表示算法的渐进复杂度，通常用于更精确的复杂度比较。

Θ符号：表示算法的平均时间复杂度，描述其在实际操作中的表现。

7.算法的五个基本属性

有穷性：算法必须在有限的步骤内终止。

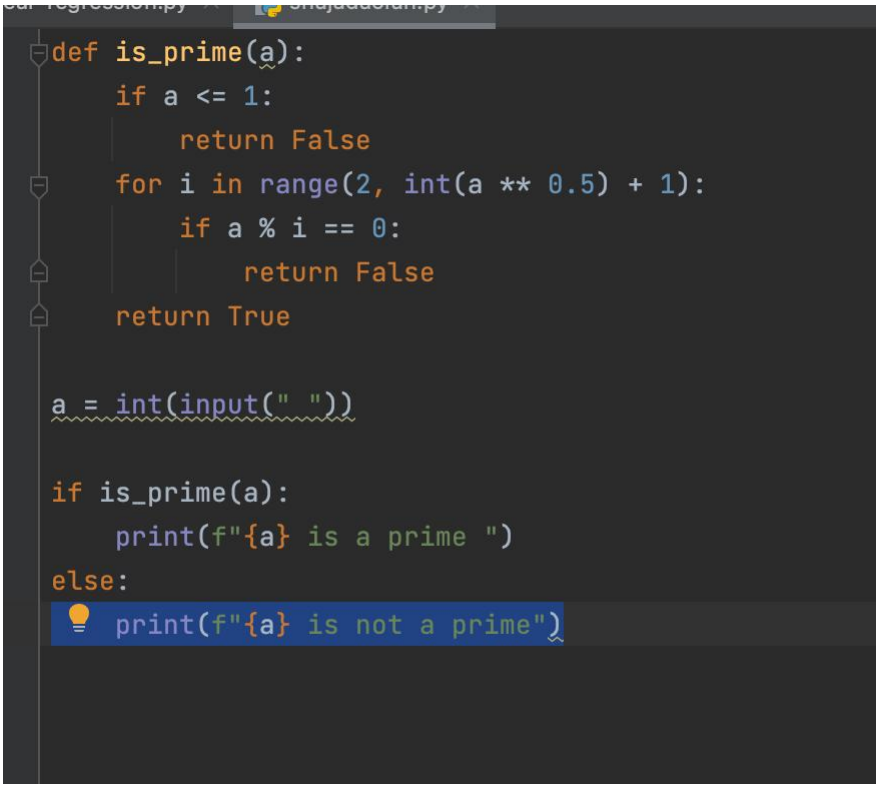
确定性：算法的每一步都有确定的含义，不会有歧义。

输入：算法应有零个或多个外部输入。

输出：算法应有一个或多个输出，反映问题的解决结果。

可行性：算法中的每一个步骤都可以有效执行，且在实际情况下可行。

践习题



```python
def is_prime(a):
    if a <= 1:
        return False
    for i in range(2, int(a ** 0.5) + 1):
        if a % i == 0:
            return False
    return True

a = int(input(" "))

if is_prime(a):
    print(f"{a} is a prime ")
else:
    print(f"{a} is not a prime")
```

6

```python
import random
import time

def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]

def test_selection_sort():
    for length in [100, 1000, 5000]:
        arr = [random.randint(0, 10000) for _ in range(length)]
        start_time = time.time()
        selection_sort(arr)
        end_time = time.time()
        print(f"数组长度: {length}, 排序时间: {end_time - start_time:.6f} 秒")
```

```
数组长度: 100, 排序时间: 0.000410 秒
数组长度: 1000, 排序时间: 0.034862 秒
数组长度: 5000, 排序时间: 0.623365 秒
```

7.

```python
def hanuota(n, source, target, path):
    if n == 1:
        print(f"移动圆盘 1 从 {source} 到 {target}")
        return
    hanuota(n-1, source, path, target)
    print(f"移动圆盘 {n} 从 {source} 到 {target}")
    hanuota(n-1, path, target, source)

n = 3
hanuota(n, 'A', 'C', 'B')
```

```
/Users/kerwinlv/anaconc
移动圆盘 1 从 A 到 C
移动圆盘 2 从 A 到 B
移动圆盘 1 从 C 到 B
移动圆盘 3 从 A 到 C
移动圆盘 1 从 B 到 A
移动圆盘 2 从 B 到 C
移动圆盘 1 从 A 到 C
```

8.

```python
def insert(root, value):
    if root is None:
        return TreeNode(value)
    if value < root.value:
        root.left = insert(root.left, value)
    else:
        root.right = insert(root.right, value)
    return root


def inorder_traversal(root, result):
    if root is not None:
        inorder_traversal(root.left, result)
        result.append(root.value)
        inorder_traversal(root.right, result)


def tree_sort(arr):
    if len(arr) == 0:
        return []

    root = None
    for value in arr:
        root = insert(root, value)

    sorted_arr = []
    inorder_traversal(root, sorted_arr)
    return sorted_arr
```

```
/Users/kerwinlv/anaconda3/envs/dzt/bin/python /Users/kerwinlv/pythonC
原始数组: [2, 15, 6, 7, 10, 1, 45, 67, 11, 23, 57, 8, 12]
排序后数组: [1, 2, 6, 7, 8, 10, 11, 12, 15, 23, 45, 57, 67]

Process finished with exit code 0
```