```
#1.
import random

def get_user_guess(lower_bound, upper_bound):
"""获取用户输入的数字，并进行有效性检查"""
while True:
try:
guess = int(input(f"请输入你的猜测 ({lower_bound} 到 {upper_bound}): "))
if lower_bound <= guess <= upper_bound:
return guess
else:
print(f"猜测必须在 {lower_bound} 到 {upper_bound} 之间。请再试一次。")
except ValueError:
print("无效输入！请输入一个数字。")

def compare_guess(guess, secret_number):
"""比较用户猜测与随机数之间的关系"""
if guess < secret_number:
return "太小了"
elif guess > secret_number:
return "太大了"
else:
return "恭喜你，猜对了！"

def T1():
"""猜数字游戏主函数"""
lower_bound = 1
upper_bound = 10
secret_number = random.randint(lower_bound, upper_bound)

print("欢迎来到猜数字游戏！")
print(f"请猜测一个 {lower_bound} 到 {upper_bound} 之间的随机数字。")

guess = get_user_guess(lower_bound, upper_bound)

result = compare_guess(guess, secret_number)
print(result)

if result == "恭喜你，猜对了！":
print(f"正确答案就是 {secret_number}。")
else:
print(f"游戏结束，正确答案是 {secret_number}。")



#2.
```

```
import random


def get_valid_guess(lower_bound, upper_bound):
"""获取有效的用户输入并验证范围"""
while True:
try:
guess = int(input(f"请输入一个 {lower_bound} 到 {upper_bound} 之间的数字: "))
if lower_bound <= guess <= upper_bound:
return guess
else:
print(f"输入的数字必须在 {lower_bound} 和 {upper_bound} 之间, 请重新输入。")
except ValueError:
print("无效输入! 请输入一个有效的数字。")


def evaluate_guess(guess, secret_number):
"""根据用户猜测与随机数字的比较结果返回提示信息"""
if guess < secret_number:
return "太小了"
elif guess > secret_number:
return "太大了"
else:
return "恭喜你, 猜对了! "


def T2():
"""游戏主流程"""
lower_bound = 1
upper_bound = 100
secret_number = random.randint(lower_bound, upper_bound)

print(f"欢迎来到猜数字游戏! ")
print(f"请选择了一个 {lower_bound} 到 {upper_bound} 之间的随机数字。")

attempts = 0

while True:
guess = get_valid_guess(lower_bound, upper_bound) # 获取有效输入
attempts += 1

result = evaluate_guess(guess, secret_number) # 评估猜测结果
print(result)

if result == "恭喜你, 猜对了! ":
```

```
print(f"你在 {attempts} 次尝试中猜对了数字 {secret_number}! ")
break

print(f"游戏结束, 正确答案是 {secret_number}。")
#3
#3.1
def generate_fibonacci(limit):
sequence = []
num1, num2 = 0, 1

while num1 < limit:
sequence.append(num1)
num1, num2 = num2, num1 + num2

return sequence

#T3.2
def find_primes_up_to(n):
sieve = [True] * n
sieve[0] = sieve[1] = False
primes = []

for number in range(2, n):
if sieve[number]:
primes.append(number)
# Mark multiples as non-prime
for multiple in range(number * number, n, number):
sieve[multiple] = False

return primes

#T3.3
def check_palindrome(input_str):
# 预处理输入字符串, 移除空格并转为小写
formatted_str = ''.join(input_str.lower().split())

# 使用递归或反转字符串检查是否是回文
return formatted_str == formatted_str[::-1]

#T3.4
import random

def create_random_list(size, min_value=0, max_value=10):
numbers = [random.randint(min_value, max_value) for _ in range(size)]
avg = sum(numbers) / len(numbers)
```

```
above_avg = list(filter(lambda x: x > avg, numbers)) # 使用filter()筛选
return numbers, avg, above_avg
#T3.5
import math

def calculate_work_effort(target_ratio=37.78, days_per_year=365,
    work_days_per_week=5, rest_days_per_week=2):
# 计算一年的工作天数和休息天数
total_work_days = (days_per_year // (work_days_per_week + rest_days_per_week)) *
    work_days_per_week
total_rest_days = days_per_year - total_work_days

# 二分法寻找最优的努力水平
low, high = 1.0, target_ratio
precision = 1e-6

while high - low > precision:
mid = (low + high) / 2
# 计算工作效率对数值
final_level_log = total_work_days * math.log(mid) + total_rest_days *
    math.log(0.99)

if final_level_log < math.log(target_ratio):
low = mid
else:
high = mid

return (low + high) / 2

#4
def find_max_overlap(str1, str2):
"""
计算并返回两个字符串之间的最大重叠长度
"""
max_overlap_length = 0
len_str1, len_str2 = len(str1), len(str2)

for overlap in range(1, min(len_str1, len_str2) + 1):
# 判断 str1 的后缀和 str2 的前缀是否相等
if str1[-overlap:] == str2[:overlap]:
max_overlap_length = overlap
return max_overlap_length
def combine_strings_with_overlap(str1, str2):
"""
合并两个字符串，去除它们之间的最大重叠部分。
```

```
返回合并后的字符串。
"""
overlap_length = find_max_overlap(str1, str2)
# 合并时去除 str2 的重叠部分
combined_str = str1 + str2[overlap_length:]
return combined_str
def get_input():
"""
获取用户输入的两个字符串。
"""
str1 = input("请输入第一个字符串：")
str2 = input("请输入第二个字符串：")
return str1, str2


def display_result(combined_str):
"""
打印最终合并后的字符串。
"""
print(f"合并后的字符串为：{combined_str}")


def T4():
"""
主程序函数：获取输入、合并字符串并显示结果。
"""
str1, str2 = get_input() # 获取输入
result = combine_strings_with_overlap(str1, str2) # 合并字符串
display_result(result) # 输出结果
```

```
import random
import string
#T1.1
def generate_random_string(length):
  """生成指定长度的随机字符串"""
  temp= ''.join(random.choice(string.ascii_letters) for _ in range(length))
  print("生成的随机字符串为:")
  print(temp)
  return temp
def count_character_frequencies(random_string):
  """统计字符频率"""
  freq_dict = {}
  for char in random_string:
  freq_dict[char] = freq_dict.get(char, 0) + 1
  return freq_dict

def display_frequency(freq_dict):
  """显示字符频率"""
  for char, count in freq_dict.items():
  print(f"{char}: {count}")

#T1.2
import random
import string

# 生成包含 1000 个随机字符的字符串
random_string = ''.join(random.choice(string.ascii_letters) for _ in range(1000))

# 使用字典推导式统计每个字符出现的次数
char_count = {char: random_string.count(char) for char in set(random_string)}

# 输出结果
print("字符出现次数: ")
for char, count in char_count.items():
print(f"{char}: {count}")

#T2.1
class SetOperations:
  def __init__(self, elements):
    self.elements = set(elements)

  def intersection(self, other):
    """计算交集"""
    return self.elements & other.elements
    def difference(self, other):
        """计算差集"""
        return self.elements - other.elements

    def union(self, other):
        """计算并集"""
        return self.elements | other.elements


    def get_set_input(name):
        """输入集合元素并返回集合"""
        elements = input(f"请输入集合{name}的元素（用空格分隔）：")
        return SetOperations(elements.split()) # 直接使用字符串分割，不做类型转换

#T2.2
def set_operations(set_a, set_b):
    # 计算交集、差集和并集
    intersection = set_a & set_b # 交集
    difference_a_b = set_a - set_b # A - B 差集
    difference_b_a = set_b - set_a # B - A 差集
    union = set_a | set_b # 并集

    # 返回结果作为字典
    return {
    "交集": intersection,
    "A - B 差集": difference_a_b,
    "B - A 差集": difference_b_a,
    "并集": union
    }


def get_set_input(name):
    """输入集合元素并返回集合"""
    elements = input(f"请输入集合{name}的元素（用空格分隔）：")
    return set(elements.split()) # 将输入的字符串分割成列表，然后转为集合

#T3

import random
import string


def generate_random_strings(n, m):
    """生成随机字符串列表"""
    return [''.join(random.choices(string.ascii_letters + string.digits,
        k=random.randint(1, m))) for _ in range(n)]


def sort_nested_lists(nested_list):
   """排序嵌套列表中的每个子列表，按字符串长度降序"""
   return [sorted(sublist, key=len, reverse=True) for sublist in nested_list]


def generate_nested_list(n, m):
   """生成嵌套列表"""
   return [generate_random_strings(random.randint(1, m), m) for _ in range(n)]


#T5
import random

   def generate_odd_tuple(n, m):
   """生成一个包含奇数的元组"""
   return tuple(num for num in (random.randint(1, m) for _ in range(n)) if num %
       2 != 0)


#T4
import random

def generate_integer_list(size, lower_bound=0, upper_bound=100):
   """生成包含指定数量的随机整数的列表"""
   return [random.randint(lower_bound, upper_bound) for _ in range(size)]

def create_lists(original_list):
   """创建新列表、逆序列表及偶数索引列表"""
   return (
   original_list.copy(), # 新列表
   original_list[::-1], # 逆序列表
   original_list[::2] # 偶数位置的元素列表
   )


#T6
def count_words_in_string(input_string):
   """统计字符串中单词出现的次数"""
   words = input_string.lower().split()
   word_count = {}
   for word in words:
   word_count[word] = word_count.get(word, 0) + 1
```

```
    return word_count



def display_word_count(word_count):
    """显示单词的出现次数"""
    for word, count in word_count.items():
    print(f"{word}: {count}")



#T7
import re

def find_three_letter_words(text):
    # 使用正则表达式找出所有长度为 3 的字母单词
    pattern = r'\b[a-zA-Z]{3}\b' # \b 表示单词边界, [a-zA-Z]
        表示字母, {3}表示3个字符
    words = re.findall(pattern, text)
    return words
```

```python
import json
#T1.1
class Student:
    def __init__(self, student_id, name, gender, age, scores):
        self.student_id = student_id
        self.name = name
        self.gender = gender
        self.age = age
        self.scores = scores # 字典存储课程成绩
        self.avg_score = 0 # 初始没有平均分

    def average_score(self):
        """计算学生的平均成绩"""
        return sum(self.scores.values()) / len(self.scores)

    def display_info(self):
        """显示学生的详细信息"""
        scores_str = ', '.join(f"{k}: {v}" for k, v in self.scores.items())
        return
            f"{self.student_id}\t{self.name}\t{self.gender}\t{self.age}\t{scores_str}"

class StudentManager:
    def __init__(self, filename):
        self.filename = filename
        self.students = self.load_students()

    def load_students(self):
        """从 JSON 文件中加载学生数据"""
        with open(self.filename, 'r', encoding='utf-8') as file:
        students_data = json.load(file)
        return [Student(**data) for data in students_data]

    def calculate_avg_scores(self):
        """为每个学生计算平均成绩"""
        for student in self.students:
        student.avg_score = student.average_score()

    def sort_by_avg_score(self):
        """按平均成绩排序"""
        self.students.sort(key=lambda s: s.avg_score)

    def assign_ranks(self):
        """为学生分配排名"""
        # 计算并分配排名
        self.sort_by_avg_score() # 确保按平均成绩排序
        rank = 1
        for student in self.students:
        student.rank = rank
        rank += 1

    def display_students(self):
        """展示所有学生的信息"""
        print("学号\t 姓名\t 性别\t 年龄\t 成绩\t 平均分\t 排名")
        for student in self.students:
        print(f"{student.display_info()}\t{student.avg_score:.2f}\t{student.rank}")

    def main_student_info():
        student_manager = StudentManager('students.json')
        student_manager.calculate_avg_scores()
        student_manager.assign_ranks() # 分配排名
        student_manager.display_students()

#T1.2
# 雇员基类
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name
        self.emp_id = emp_id
        self.salary = salary

    def pay(self):
        """基础月薪计算"""
        return self.salary

    def show(self):
        """显示员工的基本信息"""
        return f"姓名: {self.name}, 编号: {self.emp_id}, 月薪: {self.salary}"

# 经理类（继承自Employee）
class Manager(Employee):
    def __init__(self, name, emp_id, salary, department):
        super().__init__(name, emp_id, salary)
        self.department = department

    def pay(self):
        """经理薪水是基本薪水的 120%"""
        return self.salary * 1.2

    def show(self):
        """显示经理的详细信息"""
        return f"经理 - 姓名: {self.name}, 编号: {self.emp_id}, 月薪:
            {self.salary}, 部门: {self.department}"

# 销售员类（继承自Employee）
class Salesman(Employee):
    def __init__(self, name, emp_id, salary, sales):
        super().__init__(name, emp_id, salary)
        self.sales = sales

    def pay(self):
        """销售员薪水 = 基础薪水 + 销售提成"""
        return self.salary + self.sales * 0.05

    def show(self):
        """显示销售员的详细信息"""
        return f"销售员 - 姓名: {self.name}, 编号: {self.emp_id}, 月薪:
            {self.salary}, 销售额: {self.sales}"

#T1.3
# 基类: 汽车类（Vehicle）
class Vehicle:
    def __init__(self, max_speed, weight):
        self._max_speed = max_speed # 私有实例属性
        self._weight = weight    # 私有实例属性

    @property
    def max_speed(self):
        """获取最大速度"""
        return self._max_speed

    @max_speed.setter
    def max_speed(self, value):
        """设置最大速度"""
        if value > 0:
        self._max_speed = value
        else:
        print("最大速度必须是正数！")

    @property
    def weight(self):
        """获取重量"""
        return self._weight

    @weight.setter
    def weight(self, value):
        """设置重量"""
        if value > 0:
        self._weight = value
        else:
        print("重量必须是正数！")


# 派生类：自行车类（Bicycle）
class Bicycle(Vehicle):
   def __init__(self, max_speed, weight, height):
      super().__init__(max_speed, weight) # 调用父类构造函数
      self._height = height          # 自行车特有的属性：高度

   @property
   def height(self):
      """获取高度"""
      return self._height

   @height.setter
   def height(self, value):
      """设置高度"""
      if value > 0:
      self._height = value
      else:
      print("高度必须是正数！")

   @height.deleter
   def height(self):
      """删除高度属性"""
      print("删除高度属性")
      del self._height

   def set_max_speed(self, speed):
      """设置父类的最大速度"""
      self.max_speed = speed



#T1.4
class MyQueue:
def __init__(self, size):
   """初始化队列"""
   self.size = size # 队列最大长度
   self.data = [] # 队列中存储的数据
   self.current = 0 # 队列中当前的元素个数
```

```
def is_empty(self):
    """判断队列是否为空"""
    return self.current == 0

def is_full(self):
    """判断队列是否为满"""
    return self.current == self.size

def front(self):
    """获取队头元素"""
    if not self.is_empty():
    return self.data[0]
    else:
    print("队列为空，无法获取队头元素")
    return None

def enqueue(self, item):
    """将元素入队"""
    if not self.is_full():
    self.data.append(item)
    self.current += 1
    else:
    print("队列已满，无法入队")

def dequeue(self):
    """将队头元素出队"""
    if not self.is_empty():
    item = self.data.pop(0) # 删除队头元素
    self.current -= 1
    return item
    else:
    print("队列为空，无法出队")
    return None

def display(self):
    """显示队列中的所有元素"""
    if self.is_empty():
    print("队列为空")
    else:
    print("队列中的元素：", self.data)

#T2.1
def write_strings_to_file(filename, strings):
    """将多个字符串写入文件"""
    with open(filename, 'w', encoding='utf-8') as f:
    for string in strings:
    f.write(string + '\n') # 每个字符串后面加换行符

def read_and_count_strings(filename):
    """读取文件并统计字符串的个数"""
    with open(filename, 'r', encoding='utf-8') as f:
    lines = f.readlines()
    return len(lines), [line.strip() for line in lines] # 返回字符串的个数和列表

#T2.2
import re

def check_password_validity(password):
    """检查密码的有效性"""
    # 检查密码的长度
    if not (6 <= len(password) <= 12):
    return False

    # 检查密码包含至少一个小写字母
    if not re.search(r'[a-z]', password):
    return False

    # 检查密码包含至少一个数字
    if not re.search(r'[0-9]', password):
    return False

    # 检查密码包含至少一个大写字母
    if not re.search(r'[A-Z]', password):
    return False

    # 检查密码包含至少一个特殊字符
    if not re.search(r'[$#@]', password):
    return False

    # 如果通过所有检查
    return True

def filter_valid_passwords(passwords):
    """过滤有效的密码"""
    valid_passwords = []
    for password in passwords:
    if check_password_validity(password):
    valid_passwords.append(password)
    return valid_passwords
```