lecture4

1 分支结构和循环结构复习

1.1 分支结构

知识要点：

- `if` 语句：当条件为真时，执行相应的代码块。
- `elif` 语句：当前面的条件为假，且这个条件为真时，执行相应的代码块。
- `else` 语句：当前面的所有条件都为假时，执行相应的代码块。

```
grade = 85

if grade >= 90:
    print("A")
elif grade >= 80:
    print("B")
elif grade >= 70:
    print("C")
elif grade >= 60:
    print("D")
else:
    print("F")
```

1.2 循环结构

知识要点：

- `for` 循环：根据给定的序列（如列表、元组、字符串等），依次执行代码块。
- `while` 循环：当条件为真时，持续执行代码块。
- `break` 语句：跳出当前循环。
- `continue` 语句：跳过当前循环的剩余代码，进入下一次循环。

代码示例：

1. `for` 循环示例：

```python
numbers = [1, 2, 3, 4, 5]

for num in numbers:
    print(num)
```

2. `while` 循环示例：

```python
count = 1

while count <= 5:
    print(count)
    count += 1
```

3. `break` 和 `continue` 示例：

```python
for num in range(1, 11):
    if num == 6:
        break
    print(num)

for num in range(1, 11):
    if num % 2 == 0:
        continue
    print(num)
```

2 基础知识

2.1 标识符

1. **only**字母数字下划线，第一个必须为字母or下划线开头
2. 对大小写敏感（可辨认）
3. keywords.kwlist(import keyword)来看python的保留字符

2.2 字符串函数

1. +连接，*重复输出，== (! =) 比较是否相等 (不等)，[:]索引，in, not in,r或R 取消转义 (eg. Print(rn'),输出\n, %格式化输出
2. %s 字符串,%d整数，%f浮点数(%.nf保留n位小数)，%g科学计数法
3.

```python
text = "Hello, Python!"

# Get the length of the string
length = len(text)
print("Length of the string:", length)

# Convert the string to lowercase
lowercase_text = text.lower()
print("Lowercase text:", lowercase_text)

# Convert the string to uppercase
uppercase_text = text.upper()
print("Uppercase text:", uppercase_text)

# Capitalize the string
capitalized_text = text.capitalize()
print("Capitalized text:", capitalized_text)

# Replace a substring
replaced_text = text.replace("Python", "World")
print("Replaced text:", replaced_text)

# Strip leading and trailing whitespace
stripped_text = "  Hello, Python!  ".strip()
print("Stripped text:", stripped_text)

# Split the string
words = text.split()
print("Words:", words)
```

```
30
31 # Join the words back into a string
32 joined_text = " ".join(words)
33 print("Joined text:", joined_text)
```

2.3 数学运算函数

```
1  import math
2
3  x = 4
4
5  # Calculate the square root of x
6  sqrt_x = math.sqrt(x)
7  print("Square root of", x, ":", sqrt_x)
8
9  # Calculate x raised to the power of 2
10 pow_x = math.pow(x, 2)
11 print(x, "to the power of 2:", pow_x)
12
13 # Calculate the ceiling of x
14 ceil_x = math.ceil(4.3)
15 print("Ceiling of 4.3:", ceil_x)
16
17 # Calculate the floor of x
18 floor_x = math.floor(4.3)
19 print("Floor of 4.3:", floor
```

2.4 类型转换

类型转换是指将一种数据类型转换为另一种数据类型。在Python中，有一些内置函数可以实现基本类型之间的转换。

1. `int(x)`：将 x 转换为整数。如果 x 是浮点数，则向零取整；如果 x 是字符串，必须是表示整数的字符串，否则会报错。
2. `float(x)`：将 x 转换为浮点数。如果 x 是整数，则转换为浮点数；如果 x 是字符串，必须是表示浮点数或整数的字符串，否则会报错。

3. `str(x)`: 将 x 转换为字符串。这适用于整数、浮点数和其他数据类型。

```python
# Convert a float to an integer
float_num = 4.7
int_num = int(float_num)
print("Integer:", int_num)

# Convert an integer to a float
int_num = 3
float_num = float(int_num)
print("Float:", float_num)

# Convert a number to a string
num = 42
string_num = str(num)
print("String:", string_num)
```

2.5 强制类型转换

强制类型转换是指通过编程语言的特定语法或函数将一种数据类型转换为另一种数据类型。在Python中，强制类型转换通常通过类型转换函数（如上面介绍的`int()`, `float()`, `str()` 等）来实现。

注意：在强制类型转换时，需要确保转换是有意义的且不会导致数据丢失或错误。例如，将浮点数转换为整数时，小数部分将被丢弃；将非数字字符串转换为数字时，将引发错误。

```python
# Forcefully convert a float to an integer (truncates the decimal part)
float_num = 4.7
int_num = int(float_num)
print("Forcefully converted integer:", int_num)

# Forcefully convert a string to an integer (raises an error if not a valid number)
string_num = "42"
int_num = int(string_num)
print("Forcefully converted integer:", int_num)
```

```
10
11 # Forcefully convert a string to a float (raises an error if not a
   valid number)
12 string_num = "3.14"
13 float_num = float(string_num)
14 print("Forcefully converted float:", float_num)
```

需要在写程序前确保在强制类型转换之前验证输入数据的有效性。

3 data structure

Python 提供了几种内置的数据结构，用于存储和操作数据。以下是常用的基础数据结构：

3.1 列表 (List)

列表是有序的，可变的数据结构，可以容纳多个数据项。列表中的数据项可以是不同的数据类型。

key：

- 创建列表：`my_list = [1, 2, 3]`
- 访问元素：`my_list[0]`（索引从0开始）
- 切片：`my_list[1:3]`
- 添加元素：`my_list.append(4)`
- 删除元素：`del my_list[0]`
- 列表长度：`len(my_list)`
- 列表合并：`new_list = my_list1 + my_list2`
- 列表排序：`sort(reverse=True)`

注意拷贝和浅拷贝（一个是完全一样，一个只是复制了内容）。

3.2 元组 (Tuple)

元组是有序的，不可变的数据结构，可以容纳多个数据项。元组中的数据项可以是不同的数据类型。

key：

- 创建元组：`my_tuple = (1, 2, 3)`
- 访问元素：`my_tuple[0]`（索引从0开始）
- 切片：`my_tuple[1:3]`
- 元组长度：`len(my_tuple)`
- 元组合并：`new_tuple = my_tuple1 + my_tuple2`

不能修改元素值或增减元素，只能+ 合并元组。

3.3 字典 (Dictionary)

字典是无序的，可变的数据结构，用于存储键值对。字典中的键必须是唯一的，且不可变的数据类型（如字符串、数字、元组等）。

key：

- 创建字典：`my_dict = {'key1': 'value1', 'key2': 'value2'}`
- 访问元素：`my_dict['key1']`
- 添加元素：`my_dict['key3'] = 'value3'`
- 删除元素：`del my_dict['key1']`
- 检查键是否存在：`'key1' in my_dict`
- 获取键列表：`list(my_dict.keys())`
- 获取值列表：`list(my_dict.values())`

3.4 集合 (Set)

集合是无序的，可变的数据结构，用于存储唯一的数据项。（会自动去重）

key：

- 创建集合：`my_set = {1, 2, 3}` 或 `my_set = set([1, 2, 3])`

- 添加元素：`my_set.add(4)`
- 删除元素：`my_set.remove(1)`
- 检查元素是否存在：`1 in my_set`
- 集合长度：`len(my_set)`
- 集合交集：`intersection = set1 & set2`
- 集合并集：`union = set1 | set2`
- 集合差集：`difference = set1 - set2`

3.5 互相转换

大多数情况下，这些数据结构可以互相转换。以下是一些常见的转换方法：

1. **列表与元组**：
    - 列表转元组：`tuple(my_list)`
    - 元组转列表：`list(my_tuple)`
2. **列表与集合**：
    - 列表转集合：`set(my_list)`（可能会丢失重复元素和顺序）
    - 集合转列表：`list(my_set)`（转换后的列表顺序可能与原集合不同）
3. **元组与集合**：
    - 元组转集合：`set(my_tuple)`（可能会丢失重复元素和顺序）
    - 集合转元组：`tuple(my_set)`（转换后的元组顺序可能与原集合不同）
4. **字典与其他数据结构**： 字典与其他数据结构之间的转换通常涉及字典的键和值。
    - 字典的键转列表：`list(my_dict.keys())`
    - 字典的值转列表：`list(my_dict.values())`
    - 字典的键转集合：`set(my_dict.keys())`
    - 字典的值转集合：`set(my_dict.values())`
    - 字典的键值对转元组列表：`list(my_dict.items())`

4 基础算法

在本复习中，我们将讨论以下几种基础算法：

1. 汉诺塔（递归问题）
2. 蒙特卡洛方法
3. 二分查找
4. 归并排序

4.1 汉诺塔（递归问题）

汉诺塔问题是一个经典的递归问题，它涉及将一堆圆盘从一个柱子移动到另一个柱子，但在移动过程中需要遵循以下规则：

- 每次只能移动一个圆盘
- 圆盘必须从一个柱子移到另一个柱子
- 任何时候都不能将一个较大的圆盘放在较小的圆盘上面

```python
def hanoi_tower(n, source, target, auxiliary):
    if n > 0:
        hanoi_tower(n-1, source, auxiliary, target)
        print(f'Move disk {n} from {source} to {target}')
        hanoi_tower(n-1, auxiliary, target, source)

hanoi_tower(3, 'A', 'C', 'B')
```

4.2 蒙特卡洛方法

蒙特卡洛方法是一种随机抽样技术，用于近似复杂数学问题的解。以下是一个使用蒙特卡洛方法计算圆周率的例子：

```python
import random

def monte_carlo_pi(num_samples):
    inside_circle = 0

    for _ in range(num_samples):
        x = random.random()
```

```
        y = random.random()
        distance = x**2 + y**2

        if distance <= 1:
            inside_circle += 1

    return 4 * inside_circle / num_samples

print(monte_carlo_pi(1000000))
```

4.3 二分查找

二分查找是一种在有序列表中查找特定元素的算法。它每次比较列表中间的元素，直到找到目标元素或者搜索范围为空。

```
def binary_search(arr, target):
    low, high = 0, len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1

    return -1

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(binary_search(arr, 5))
```

4.4 归并排序

归并排序是一种分治算法，它将一个大问题分成两个或多个较小的问题来解决，然后将这些较小问题的解合并以得到原问题的解。

```
def merge_sort(arr):

```

```
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])

    return merge(left, right)

def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])

    return result

arr = [38, 27, 43, 3, 9, 82, 10]
print(merge_sort(arr))
```