DS 부트 캠프 2024 Day2(v1.5)

한상곤(Sangkon Han)
부산대학교 정보컴퓨터공학부
2024/02/06

목차

함수와 매개변수

함수 기초

- 함수(function)
  - 어떤 일을 수행하는 코드의 덩어리, 또는 코드의 묶음
- 함수의 장점
  - 논리적인 단위로 분할 가능
  - 코드의 캡슐화

함수 선언 (1)

```
def 함수 이름 (매개변수 #1 ...):
    명령문 1
    명령문 2
    return <반환값>
```

- `def`: 'definition'의 줄임말로 함수의 정의를 시작
- `함수 이름`:
  - 소문자 입력
  - 띄어쓰기를 할 경우에는 `_` 기호 사용
  - 작업을 나타내기 위해 동사와 명사를 함께 사용하는 경우가 많음
  - 외부에 공개하는 함수일 경우 줄임말을 사용하지 않고 짧고 명료한 이름으로 정함

함수 선언 (2)

- 매개변수(parameter)
  - 매개변수는 함수에서 입력값으로 사용하는 변수를 의미하며, 1개 이상의 값을 적을 수 있음
- 명령문
  - 명령문은 반드시 들여쓰기한 후 코드를 입력해야 함
- 반환값
  - 명령문 실행 후 결과값을 전달
- 매개변수/반환값의 유/무에 따라 함수의 작성 및 방향성이 달라짐을 주의

함수 호출

```
def f(x):
    y = x
    x = 5
    return y * y

x = 3
print(f(x)) # 9
print(x) # 3
```

- `3행`과 `6행`에서 함수 `f(x)`의 `x`에 `5`와 `3`이 할당
- 함수 안에서의 x와 함수 밖에서의 x는 같은 변수일까, 아니면 다른 변수일까?
  - if 문을 사용할 때 키워드 is가 변수들의 메모리 주소를 비교함. 즉, 함수 밖에 있는 변수 x의 메모리 주소와 함수 안에 있는 변수 x의 메모리 주소가 같은지, 다른지 확인해야 함.

함수 호출 방식

- 메모리 주소: 변수가 저장되는 공간으로 그 공간 자체에 새로운 값을 할당하면 그 공간을 가리키고 있는 다른 변수에도 영향을 줌
  - 만약 참조 호출로 적용된다면 맨 마지막에 있는 `x`의 값은 `5`로 변환되어야 하지만 파이썬은 객체의 주소가 함수로 넘어간다는 뜻의 `객체 호출(call by object reference) 방식`을 사용함

```
def spam(eggs):
    eggs.append(1)   # 기존 객체의 주소값에 [1] 추가
    eggs = [2, 3]    # 새로운 객체 생성
ham = [0]
spam(ham)
print(ham)
```

변수의 사용 범위

- 변수의 사용 범위(scoping rule)
  - 변수가 코드에서 사용되는 범위
- 변수의 사용 범위를 결정할 때 고려해야 할 두 가지 변수
  - 지역 변수(local variable)
    - 함수 내부에서만 사용
  - 전역 변수(global variable)
    - 프로그램 전체에서 사용

변수의 사용 범위 (예)

```
def test(t):
    print(x)
    t = 20
    print("In Function:", t)
x = 10
test(x)
print("In Main:", x)
print("In Main:", t)
```

함수의 인수

- 함수의 인수(argument)
  - 함수의 입력으로 들어가는 변수의 다양한 형태
- 종류
  - 키워드 인수
  - 디폴트 인수
  - 가변 인수
  - 키워드 가변 인수

함수의 인수 - 키워드 인수

```
def print_something(my_name, your_name):
    print("Hello {0}, My name is {1}".format(your_name, my_name))
print_something("홍길동", "유관순")
print_something(your_name = "홍길동", my_name = "유관순")
```

함수의 인수 - 디폴트 인수

```
def print_something_2(my_name, your_name = "유관순"):
    print("Hello {0}, My name is {1}".format(your_name, my_name))
print_something_2("홍길동", "유관순")
print_something_2("홍길동")
```

함수의 인수 - 가변 인수

```
def asterisk_test(a, b, *args):
    print(args)
    return a + b + sum(args)
print(asterisk_test(1, 2, 3, 4, 5))
```

```
def asterisk_test2(*args):
    x, y, *z = args
    return x, y, z
print(asterisk_test(3, 4, 5))
```

함수의 인수 - 키워드 가변 인수

```
def kwargs_test(**kwargs):
    print(kwargs)
    print("First value is {first}".format(**kwargs))
    print("Second value is {second}".format(**kwargs))
    print("Third value is {third}".format(**kwargs))
kwargs_test(first = 3, second = 4, third = 5)
```

함수 연습

- 버블 정렬
- 선택 정렬
- 삽입 정렬
- 퀵 정렬

```
# Write your code
```

재귀

재귀 (예)

- 재귀 함수(recursive function)
  - 자기 자신을 다시 호출하는 함수
  - 재귀적이라는 표현은 자신을 이용해 다른 것을 정의한다는 뜻

```
def fib_rec(n):
    if n < 3: return 1
    return fib_rec(n - 1) + fib_rec(n - 2)
result = fib_rec(10)
```

재귀의 필요성

- 구현해야 할 알고리즘이 반복문보다 훨씬 자연스러울 때
  - 점화식이 존재하는 문제는 재귀 함수로 접근하는 방법이 더 쉬움
- 상태 변화시키는 데 변수를 사용할 일이 줄어 사이드 이펙트를 많이 줄일 수 있음

재귀의 필요성

- 재귀를 사용하면 실무에서 재귀를 잘 사용하지 않는 이유 중 결정적 원인인 스택 오버플로(stack overflow)가 발생할 수 있음
- 재귀 함수를 사용할 경우 호출 스택(stack)에 누적되는 구조, 즉 함수에서 함수를 실행했을 때 실행이 완료되고 다시 돌아가기 위한 주소가 스택이라는 메모리 구조에 계속해서 쌓임
  - 파이썬의 경우 1,000개가 한계
  - 계속 재귀 제한 횟수 초과 오류(RecursionError)가 발생한다면 임의로 `sys.setrecursionlimit()` 함수를 사용하여 제한 횟수를 늘릴 수 있음

꼬리 재귀(tail recursion)

- 꼬리 재귀는 기존 재귀 방식의 큰 문제점이었던 스택이 초과될 수 있다는 단점을 개선하기 위해 만들어진 개념
- return으로 함수를 반환할 때(재귀 호출을 할 때) 아무런 연산도 하지 않게 함으로써 함수 종료 이후 곧바로 결과만 반환하게 만들어 스택 사용을 줄일 수 있는 장점이 있음

```
def fib_tail(n, first, second):
    if n <= 1: return second
    return fib_tail(n - 1, second, first + second)
```

재귀와 꼬리 재귀(tail recursion)

```
a = time.time()
print(fib_rec(35))
print(time.time() - a)
a = time.time() - a

b = time.time()
print(fib_tail(35, 0, 1))
print(time.time() - b)
b = time.time() - b

print(f'TR runs x{int(a / b)} faster than normal recursion') # maybe error?
```

재귀 및 꼬리 함수 연습

- 입력으로 들어온 리스트의 값들을 뒤집는 재귀 함수
- 입력 리스트에서 입력받은 숫자만큼 값만 꺼내오는 함수
- 두 개의 리스트를 입력으로 받아서 하나의 리스트로 조합하는 함수
- 입력으로 들어온 리스트의 값들을 뒤집는 꼬리 재귀 함수
- 입력 리스트에서 입력받은 숫자만큼 값만 꺼내오는 꼬리 재귀 함수
- 두 개의 리스트를 입력으로 받아서 하나의 리스트로 조합하는 꼬리 재귀 함수

실험 퀵 정렬

- 퀵 정렬(quick sort)은 극단적인 경우를 제외하면 $O(nlogn)$ 정렬 알고리즘 중에서 가장 빠릅니다. 적절한 원소 하나를 기준 삼아(이를 피벗(pivot)이라고 합니다) 기준보다 작으면 앞으로, 크면 뒤로 빼고, 더 이상 움직일 게 없다면 기준 위치에서 배열을 나눈 다음, 다시 나눠진 배열에서 기준을 잡아 정렬이 완료될 때까지 이를 반복합니다. 분할-정복을 기반으로 하며, 재귀로 구현할 수 있습니다.

```python
def quick_sort(l):
    if len(l) <= 1:
        return l
    pivot = l[len(l) // 2]
    lesser_arr, equal_arr, greater_arr = [], [], []
    for num in l:
        if num < pivot:
            lesser_arr.append(num)
        elif num > pivot:
            greater_arr.append(num)
        else:
            equal_arr.append(num)
    return quick_sort(lesser_arr) + equal_arr + quick_sort(greater_arr)

data = [3, 0, 1, 8, 7, 2, 5, 4, 6, 9]
result = quick_sort(data)
print(result)
```

과제02 : 병합 정렬 작성

간단한 제어문 사용하여 병합 정렬 알고리즘을 구현하세요. 병합 정렬 알고리즘은 $O(nlog(n))$ 시간 복잡도 내에서 데이터를 정렬할 수 있는 효율적인 정렬 알고리즘으로, 여기서 n은 값의 개수입니다.

과제2 병합 정렬 코드 작성

- 제시된 `assignment02.py`을 사용하세요.
- 우측에 보이는 `Algorithm 1`을 참고해서 코드를 작성하세요.
- 제시된 알고리즘을 작성하시면 정렬된 결과를 확인하실 수 있습니다.

```
$ python .\assignment02.py
[2, 3, 0, 1, 5, 4, 6, 8, 7, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Algorithm 1: Merge Function

Data: A: The array to be sorted
L1: The start of the first part
R1: The end of the first part
L2: The start of the second part
R2: The end of the second part
Result: Return the merged sorted array

Function $merge(A, L1, R1, L2, R2)$**:**
    $temp \leftarrow \{\}$;
    $index \leftarrow 0$;
    while $L1 \leq R1$ ***AND*** $L2 \leq R2$ **do**
        if $A[L1] \leq A[L2]$ **then**
            $temp[index] \leftarrow A[L1]$;
            $index \leftarrow index + 1$;
            $L1 \leftarrow L1 + 1$;
        else
            $temp[index] \leftarrow A[L2]$;
            $index \leftarrow index + 1$;
            $L2 \leftarrow L2 + 1$;
        end
    end
    while $L1 \leq R1$ **do**
        $temp[index] \leftarrow A[L1]$;
        $index \leftarrow index + 1$;
        $L1 \leftarrow L1 + 1$;
    end
    while $L2 \leq R2$ **do**
        $temp[index] \leftarrow A[L2]$;
        $index \leftarrow index + 1$;
        $L2 \leftarrow L2 + 1$;
    end
    return $temp$;
end