Fastcampus Data Science Extension SCHOOL

Python

Index

- Recursive, Iterative factorial
- Comprehension
  - list comprehension
  - dictionary comprehension
- Decorator
- Programming Paradigms
  - Sequential Programming
  - Procedural Programming
  - Object Oriented Programming

variable outside function

```
a = "hello"
def glob_test(a):
        a += "world"
        return a

glob_test(a)
print(a)
```

```
a = "hello"
def glob_test(x):
        x += "world"
        return x

glob_test(a)
print(a)
```

variable outside function

```
def glob_test2(x):
    a += "world"
    x += "success"
        return x

glob_test2(a)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in glob_test2
UnboundLocalError:
local variable 'a' referenced before assignment
```

So, how to globalize

(1) using return

```
a = "hello"
def glob_test(a):
        a += "world"
    return a

a = glob_test(a)
print(a)
```

So, how to globalize

(2) use global

```
a = "hello"
def glob_test(a):
        global a
        a += "world"
    return a

glob_test(a)
print(a)
```

global 이라는 명령을 사용하여 전역변수로 사용하게 되면 함수는 독립성을 잃게 되어 함수가 외부변수에 의존적이게 됩니다.

Recursive

What is GNU?

- GNU is Not Unix
  - What is GNU?
  - GNU is Not Unix
    - What is GNU?
    - GNU is Not Unix
      - What is GNU?
      - GNU is Not Unix
        - ...

Recursive

```python
times = int(input("How many times want to curse the beast??: "))
def recurse_beast(a):
        if a == 0:
                print("curse complete!")
        else:
                print("Fusion!!!(%d times left)" % a - 1)
                recurse_beast(a-1)

recurse_beast(times)
```

Fibonacci Sequence

Fibonacci Sequence

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n > 1 \end{cases}$$

Fibonacci Sequence with Recursion

```python
def fib_rec(n):
    if n < 2:
        return n
    else:
        return fib_rec(n-1) + fib_rec(n-2)
```

Binet's Fibonacci formula

$$F_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}$$

Binet's Fibonacci formula

```python
import math


def fib_binet(n):
    sqrt_5 = math.sqrt(5)
    result = int(((1+sqrt_5)**n-(1-sqrt_5)**n) / (2**n*sqrt_5))
    return result
```

실행시간을 비교해봅시다.

- 40개의 피보나치 수 구하기
  - case 1:

    ```
    execution time: 90.98334097862244
    ```

  - case 2:

    ```
    execution time: 0.00013065338134765625
    ```

Fibonacci Recursion Flow

F_6 ->

F_5+F_4 ->

F_4+F_3 + F_3+F_2 ->

F_3+F_2 + F_2+F_1 + F_2+F_1 + F_1+F_0 ->

F_2+F_1 + F_1+F_0 + F_1+F_0 + 1 + F_1+F_0 + 1 + 1 + 0 ->

F_1+F_0+ 1 + 1+0 + 1+0 + 1 + 1+0 + 1 + 1 + 0 ->

1+0+ 1 + 1+0 + 1+0 + 1 + 1+0 + 1 + 1 + 0

= 8

Recursion 장점

- 재귀적 알고리즘 표현이 명확할 경우 Loop 사용보다 직관적인 코드
- 변수의 수를 줄이고, 가능한 경우의 수를 줄여줘 오동작 가능성이 줄어듦

Recursion 사용시 주의사항

- Python은 function depth가 1000으로 제한되며, 근접시 동작하지 않습니다.
- 시간복잡도를 감안해 Recursion을 작성해야 합니다.
- Recursion을 Escape할 장치를 마련해야 합니다.

Do it yourself!

사용자의 입력 `num` (0~950 사이의 정수)을 받아
1에서 `num` 까지의 모든 자연수의 곱(팩토리얼)을 구하는 함수를
Recursive, Iterative 두가지 방법으로 해결하세요.

Answer

Recursive

```python
def factorial_rec(num):
    if num<2:
        return 1
    else:
        return num*factorial_rec(num-1)
```

Iterative

```python
def factorial_iter(num):
    if num<2:
        return 1
    else:
        result = 1
        for i in range(1,num+1):
            result *= i
        return result
```

List Comprehension

존재하는 리스트를 활용하여 새로운 리스트를 생성하는 방법

비슷한 표현들

- Set Comprehension
- Dictionary Comprehension
- Parallel list Comprehension

List Comprehension

```
old_list = [1, 2, 3, 4, 5,]

doubled_list = []
for i in old_list:
        doubled_list.append(i * 2)
```

List Comprehension

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

doubled_list = []
for i in old_list:
        doubled_list.append(i * 2)
```

```python
doubled_list = []
```

List Comprehension

```
old_list = [1, 2, 3, 4, 5,]

doubled_list = []
for i in old_list:
        doubled_list.append(i * 2)
```

```
doubled_list = [i * 2]
```

List Comprehension

```
old_list = [1, 2, 3, 4, 5,]

doubled_list = []
for i in old_list:
        doubled_list.append(i * 2)
```

```
doubled_list = [i * 2 for i in old_list]
```

List Comprehension - another example

```
old_list = [1, 2, 3, 4, 5,]

doubled_list = []
for i in old_list:
        if i % 2 == 0:
                doubled_list.append(i * 2)
```

List Comprehension - another example

```
old_list = [1, 2, 3, 4, 5,]

doubled_list = []
for i in old_list:
        if i % 2 == 0:
                doubled_list.append(i * 2)
```

```
doubled_list = []
```

List Comprehension - another example

```
old_list = [1, 2, 3, 4, 5,]

doubled_list = []
for i in old_list:
        if i % 2 == 0:
                doubled_list.append(i * 2)
```

```
doubled_list = [i * 2]
```

List Comprehension - another example

```
old_list = [1, 2, 3, 4, 5,]

doubled_list = []
for i in old_list:
        if i % 2 == 0:
                doubled_list.append(i * 2)
```

```
doubled_list = [i * 2 for i in old_list]
```

List Comprehension - another example

```
old_list = [1, 2, 3, 4, 5,]

doubled_list = []
for i in old_list:
        if i % 2 == 0:
                doubled_list.append(i * 2)
```

```
doubled_list = [i * 2 for i in old_list if i % 2 == 0]
```

Do it yourself!

- List comprehension 으로 FizzBuzz 한줄로 구현하기

```
 ["Fizz"*(not i%3) + "Buzz"*(not i%5) or i for i in
range(1,100)]
```

Dictionary Comprehensions

Dictionary Comprehensions

- 기존의 딕셔너리를 활용해 새로운 딕셔너리를 만들고 싶을때

Dictionary Comprehensions

```python
old_dict = {1:1,2:2,3:3,4:4,}
new_dict = {}
for k,v in old_dict.items():
    new_dict[k]=v*2
```

Dictionary Comprehensions

```python
old_dict = {1:1,2:2,3:3,4:4,}
new_dict = {}
for k,v in old_dict.items():
    new_dict[k]=v*2
```

```python
new_dict = {}
```

Dictionary Comprehensions

```
old_dict = {1:1,2:2,3:3,4:4,}
new_dict = {}
for k,v in old_dict.items():
    new_dict[k]=v*2
```

```
new_dict = {k:v*2} #new_dict[k]=v*2
```

Dictionary Comprehensions

```
old_dict = {1:1,2:2,3:3,4:4,}
new_dict = {}
for k,v in old_dict.items():
    new_dict[k]=v*2
```

```
new_dict = {k:v*2 for k,v in old_dict.items()}
```

Dictionary Comprehensions

```
old_dict = {1:1,2:2,3:3,4:4,}
new_dict = {}
for k,v in old_dict.items():
    if v%2!=0:
        new_dict[k*2]=v*3
```

Dictionary Comprehensions

```
old_dict = {1:1,2:2,3:3,4:4,}
new_dict = {}
for k,v in old_dict.items():
    if v%2!=0:
        new_dict[k*2]=v*3
```

```
new_dict = {}
```

Dictionary Comprehensions

```
old_dict = {1:1,2:2,3:3,4:4,}
new_dict = {}
for k,v in old_dict.items():
    if v%2!=0:
        new_dict[k*2]=v*3
```

```
new_dict = {k*2:v*3} #new_dict[k*2]=v*3
```

Dictionary Comprehensions

```
old_dict = {1:1,2:2,3:3,4:4,}
new_dict = {}
for k,v in old_dict.items():
    if v%2!=0:
        new_dict[k*2]=v*3
```

```
new_dict = {k*2:v*3 for k,v in old_dict.items()}
```

Dictionary Comprehensions

```
old_dict = {1:1,2:2,3:3,4:4,}
new_dict = {}
for k,v in old_dict.items():
    if v%2!=0:
        new_dict[k*2]=v*3
```

```
new_dict = {k*2:v*3 for k,v in old_dict.items() if v%2!=0}
```

Decorator

Back to the Fibonacci..

```
start_time = time.time()
fib_rec(10)
end_time = time.time()
print(end_time-start_time)
```

```
start_time = time.time()
fib_binet(10)
end_time = time.time()
print(end_time-start_time)
```

Let's wrap with decorator

```python
import time


def time_checker(function):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = function(*args, **kwargs)
        end_time = time.time()
        print("execution time: {time} sec".format(
                time=end_time-start_time))
        return result
    return wrapper
```

and just add `@`

```
@time_checker
def fib_rec(num):
        ...
```

Decorator는

- function의 앞, 뒤로 해야 할 일이나 로깅, 벤치마킹 등 다양한 용도로 쓰일 수 있습니다.
- 데이터 전처리 과정 또한 미리 정의해둔 뒤, 붙여 사용할 수 있습니다.

Do it yourself!

"Hi, {name}. You might be loved with {lang}" 이라는 문자열이 존재할 때, 이 문자열의 앞 뒤로 `<h1>, <em>` 태그가 붙도록 하는 데코레이터를 생성하세요

ex output)

```
<h1><em> {{text}} </em></h1>
```

`Advanced problem: Decorator 하나로 html 태그 이름을 지정할 수 있도록 수정`

Programming Paradigms

Sequencial Programming

```
print("wake up")
print("go to work")
print("have lunch")
print("hard work")
print("back to home")
print("have dinner")
print("get some sleep")
if tomorrow == "weekend":
        goto 12
else :
        goto 1
print("zzz")
print("have dinner")
print("get some sleep")
if tomorrow == "weekend":
        goto 12
else :
        goto 1
```

Procedural Programming

```python
def wake():
    return "wake up"
def eat():
    return "eat something"
def work():
    return "work"
def sleep():
    return "sleep"

while True:
    if today=="weekday":
        wake()
        work()
        eat()
        work()
        sleep()
    else:
        sleep()
        wake()
        eat()
        sleep()
```

Object-Oriented Programming

```
Class Person:
    def __init__(self):
        self.health=100
        self.hunger=100
        self.damage=1

Class Hero:
    def __init__(self, a, b):
        self.health=a
        self.hunger=100
        self.damage=b

me = Person()
iron_man = Hero(1000,1000)
hulk = Hero(10000,800)
hawk_eye = Hero(100,300)
```