Modul 8 Praktikum Pemrograman Berbasis Fungsi

April 9, 2023

1 Recursion

Tujuan Praktikum

1. Menerapkan konsep python fungsi rekursi

The word recursion comes from the Latin word recurrere, meaning to run or hasten back, return, revert, or recur. Here are some online definitions of recursion:

Dictionary.com: The act or process of returning or running back

Wiktionary: The act of defining an object (usually a function) in terms of that object itself

The Free Dictionary: A method of defining a sequence of objects, such as an expression, function, or set, where some number of initial objects are given and each successive object is defined in terms of the preceding objects

A recursive definition is one in which the defined term appears in the definition itself. Self-referential situations often crop up in real life, even if they aren't immediately recognizable as such.

Saat Anda memanggil fungsi dengan Python, interpreter membuat ruang nama lokal baru sehingga nama yang ditentukan di dalam fungsi itu tidak bertabrakan dengan nama identik yang ditentukan di tempat lain. Satu fungsi dapat memanggil yang lain, dan bahkan jika keduanya mendefinisikan objek dengan nama yang sama, semuanya berfungsi dengan baik karena objek tersebut ada di ruang nama yang terpisah.

Hal yang sama berlaku jika beberapa instance dari fungsi yang sama berjalan secara bersamaan. Sebagai contoh, pertimbangkan definisi berikut:

```
def function():
    x = 10
    function()
```

Saat function() dijalankan pertama kali, Python membuat namespace dan memberikan x nilai 10 di namespace tersebut. Kemudian function() memanggil dirinya sendiri secara rekursif. Kedua kali function() dijalankan, interpreter membuat namespace kedua dan menetapkan 10 ke x di sana juga. Kedua contoh nama x ini berbeda satu sama lain dan dapat berjalan berdampingan tanpa bentrok karena berada di ruang nama yang terpisah.

Similarly, a function that calls itself recursively must have a plan to eventually stop. Recursive functions typically follow this pattern:

- There are one or more base cases that are directly solvable without the need for further recursion.

- Each recursive call moves the solution progressively closer to a base case.

2 Contoh

2.1 Countdown

[ ]:
```
def countdown(n):
     print(n)
     if n == 0:
         return             # Terminate recursion
     else:
         countdown(n - 1)   # Recursive call

countdown(5)
```

```
5
4
3
2
1
0
```

Notice how countdown() fits the paradigm for a recursive algorithm described above:

The base case occurs when n is zero, at which point recursion stops. In the recursive call, the argument is one less than the current value of n, so each recursion moves closer to the base case.

[ ]:
```
def countdown(n):
    print(n)
    if n > 0:
        countdown(n - 1)
```

Berikut bukan rekursif

[ ]:
```
def countdown(n):
     while n >= 0:
         print(n)
         n -= 1

countdown(5)
```

```
5
4
3
2
1
0
```

2.2 Factorial

[ ]:
```
def factorial(n):
     return 1 if n <= 1 else n * factorial(n - 1)

factorial(4)
```

[ ]:
```
24
```

[ ]:
```
def factorial(n):
     print(f"factorial() called with n = {n}")
     return_value = 1 if n <= 1 else n * factorial(n -1)
     print(f"-> factorial({n}) returns {return_value}")
     return return_value

factorial(4)
```

```
factorial() called with n = 4
factorial() called with n = 3
factorial() called with n = 2
factorial() called with n = 1
-> factorial(1) returns 1
-> factorial(2) returns 2
-> factorial(3) returns 6
-> factorial(4) returns 24
```

[ ]:
```
24
```

2.3 Eksponensial

[ ]:
```
def exponential(a, n):
    if n == 0:
        return 1
    return exponential(a, n-1) * a


print(exponential(5, 3))
```

```
125
```

2.4 Multiply

[ ]:
```
def multiply(n, a):
    if n == 1:
        return a
    return multiply(n-1, a) + a


print(multiply(5, 4))
```

```
20
```

3 Contoh lain

[ ]:
```python
def number(n):
    if(n>0):
        number(n-1)
        print(n)

number(10)
```

```
1
2
3
4
5
6
7
8
9
10
```

[ ]:
```python
def number_r(n):
    if(n>0):
        print(n)
        number_r(n-1)

number_r(10)
```

```
10
9
8
7
6
5
4
3
2
1
```

[ ]:
```python
def odd_number(n):
    if(n>0):
        odd_number(n-1)
        if(n%2!=0):
            print(n)
odd_number(10)
```

```
1
3
```

```
5
7
9
```

[ ]:
```python
def odd_number(n):
    if(n>0):
        odd_number(n-1)
        if(n%2!=0):
            print(n)
odd_number(10)
```

```
1
3
5
7
9
```

[ ]:
```python
def odd_rev(n):
    if(n>0):
        if(n%2!=0):
            print(n)
        odd_rev(n-1)


odd_rev(10)
```

```
9
7
5
3
1
```

[ ]:
```python
def even_num(n):
    if(n>0):
        even_num(n-1)
        if(n%2==0):
            print(n)

even_num(10)
```

```
2
4
6
8
10
```

[ ]:
```python
def square(n):
    if(n>0):
        square(n-1)
        print(n*n)

square(10)
```

```
1
4
9
16
25
36
49
64
81
100
```

[ ]:
```
def square_rev(n):
    if(n>0):

        square_rev(n-1)
        print(n*n*n)

square_rev(10)
```

```
1
8
27
64
125
216
343
512
729
1000
```

[ ]:
```
def multiple(n):
    for i in range(1,11,1):
        print(n*i)

multiple(10)
```

```
10
20
30
40
50
60
70
80
90
```

100