

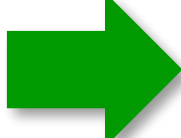
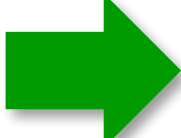
# ③

## 計算量の見積もり方

---

最後にプログラムの計算時間の見積もり方の基本を学習します。

# 計算量の見積もり方の基本

- ここでは  $n$  を引数とした関数の計算時間をオーダー記法で見積もることを考えます
- この時の基本は、関数内の決まった処理の繰り返しの回数に注目します
  - 例)  $n$  によらず繰り返し回数は一定   $O(1)$
  - 例) 繰り返し回数は  $n$  に比例   $O(n)$

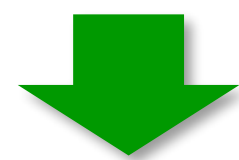
# for文を含む関数の実行時間①

- 以下のように、決まった処理がfor文で繰り返される場合は、それが何回繰り返されるかに着目し、計算量を見積もります
  - 二重ループの場合は、 $n^2$  回繰り返されます

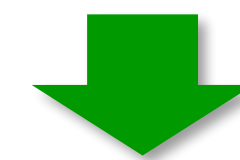
```
def func1(n):  
    s = 0  
    for i in range(n):  
        # Something to do  
        s += 1  
    return s
```

 $O(n)$ 

```
def func2(n):  
    s = 0  
    for i in range(n):  
        for j in range(n):  
            s += 1  
    return s
```

 $O(n^2)$ 

```
def func3(n):  
    s = 0  
    for i in range(2 ** n):  
        s += 1  
    return s
```

 $O(2^n)$

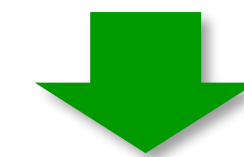
## for文を含む関数の実行時間②

- 以下のように繰り返す回数が定数である場合には、その部分の処理は一定時間で終わることに注意しましょう
  - 計算量の意味合いを考えれば、明らかですね？

```
def func4(n):  
    s = 0  
    for i in range(n):  
        for j in range(100):  
            s += 1  
    return s
```

 $O(n)$ 

```
def func5(n):  
    s = 0  
    for i in range(100):  
        for j in range(100):  
            s += 1  
    return s
```

 $O(1)$

# 例題①

- それでは、以下のような関数の計算量をオーダー記法で記述すると、どのようになりますか？

```
def func6(n):  
    s = 0  
    for i in range(n):  
        for j in range(n):  
            for k in range(n):  
                s += 1  
  
    t = 0  
    for i in range(2 ** n):  
        t += 1  
  
    return s
```

$O(n^3)$

$O(2^n)$

$\Rightarrow O(2^n)$

計算時間が複数の項の和となる場合は、その中で最も早く増加する項に着目することに注意！

## 例題②

- それでは、以下のような関数の計算量をオーダー記法で記述すると、どのようになりますか？

```
def func7(n):  
    for i in range(n):  
        for j in range(n):  
            func1(n)  
  
    for k in range(100):  
        func2(n)
```

 $O(n^3)$  $O(n^2)$  $\Rightarrow O(n^3)$ 

他の関数を繰り返し呼び出す場合は、計算量の積をとります  
(意味を考えれば、自明ですよ？)