

関数を定義する

初めてのプログラミング

2020年度

只木進一（理工学部）

サンプルプログラムの取得

- GitHubRepositoryを指定
 - <https://github.com/first-programming-saga/functions>

関数を定義する目的

- ▶ 機能毎にプログラムを作る
 - ▶ 動作確認が容易
 - ▶ 繰り返し利用
- ▶ 小さな関数を沢山書くのが良い

関数を定義する：引数の無い関数

Functions without arguments

```
def 関数名():  
    関数の実体
```

- 関数の中で、何か処理する
 - 必要に応じて結果を返す
- 結果を戻す
 - return 文
- 呼び出しは、単に関数名を使う
 - 戻り値があれば、代入する

```
functions/simpleFunctions.ipynb
```

```
def hello():  
    print("こんにちは")
```

```
def goodbye():  
    print("さようなら")
```

```
hello()  
goodbye()
```

```
def hello2():  
    return "こんにちは"
```

```
def goodbye2():  
    return "さようなら"
```

```
m1=hello2()  
m2=goodbye2()  
print(m1)  
print(m2)
```

関数呼び出し

関数を定義する:引数の有る関数

Functions with arguments

```
def 関数名(引数並び):  
    関数の実体
```

➡ 引数

➡ 関数に渡す変数

➡ 全て参照渡し

➡ int, float, str, などは元の値は変わらない

➡ mutable変数は元の値が変わる

➡ 結果を戻す

➡ return 文

functions/argumentsTest.ipynb

```
In [1]: 1 def func1(x):
2         x += 1 #関数内部で値を変更
3         return x
4
5 y = 1
6 z = func1(y)
7 print(f'y={y}, z={z}') #呼び出し元の値は変更されない
```

y=1, z=2

mutable変数(変更できる変数)が引数の場合、関数内部でその要素を変更すると、呼び出し側の要素も変化する。

```
In [2]: 1 def func2(d):
2         for i in range(len(d)):
3             x = d[i]*2
4             d[i]=x
5
6 data = [1,4,2,5]
7 func2(data)
8 print(data)
```

[2, 8, 4, 10]

```
In [4]: 1 def func4(dd):
        2     d = list(dd)
        3     for i in range(len(d)):
        4         x = d[i]*2
        5         d[i]=x
        6     return d
        7
        8 data = [1,4,2,5]
        9 data2 = func4(data)
       10 print(data)
       11 print(data2)
```

```
[1, 4, 2, 5]
[2, 8, 4, 10]
```

引数と戻り値のある関数

Functions with arguments and return values

```
def quadraticFunction(x,a,b,c):  
    y = a*x*x + b*x + c  
    return y
```

```
a = 1
```

```
b = -2
```

```
c = 1
```

```
for i in range(100):
```

```
    x = i * 0.1
```

```
    y=quadraticFunction(x,a,b,c)
```

```
    message = f'f({x})={y}'
```

```
    print(message)
```

$$f(x) = ax^2 + bx + c$$

[functions/quadraticFunction.ipynb](#)

再帰的関数

Recursive function

```
def factorial(n):  
    if n==1:  
        return 1  
    return n * factorial(n-1)
```

```
m = 5  
a = factorial(m)  
print(a)
```

再帰的に呼び出し

$$n! = \begin{cases} n \times (n-1)! & n > 1 \\ 1 & n = 1 \end{cases}$$

再帰的関数：関数が、自身で定義されている

[functions/factorial.ipynb](#)

引数と戻り値のある関数 タプルを戻す例

```
1. def isPrime(n):
2.     result = False
3.     if n <= 0:
4.         message=f'引数は正でなければならない'
5.     elif n < 2:
6.         message = f'{n}は素数ではない'
7.     elif n == 2:
8.         message = f'{n}は素数である'
9.         result = True
10.    elif n % 2 == 0:
11.        message = f'{n}は偶数であり、素数ではない'
12.    else:
13.        m=int(math.sqrt(n))
14.        for k in range(3,m+1,2):#forで記述
15.            if n % k == 0:
16.                message = f'{n}は{k}で割り切れるため、素数ではない'
17.                break
18.        else:#ループの最後まで至った場合
19.            message = f'{n}は素数である'
20.            result = True
21.    return result,message
```

引数がリストの関数

```
1. def stat(data):#dataは数値のリスト
2.     n = len(data)
3.     s = 0#和を保存
4.     for x in data:#data中のすべてに対して繰り返し
5.         s += x
6.     average = s/n#平均
7.     return n, average
8.
9. #データリスト
10. dataList = [3,5,7,2,3,6,1,4,9,2]
11. n, average = stat(dataList)
12. print('データ数:',n)
13. print('平均:',average)
```

変数の有効範囲：scope

- ➡ 関数の引数や関数内で定義された変数
 - ➡ 関数の内部だけで有効：ローカル変数
 - ➡ 別の関数に同じ名前が現れても別物
- ➡ 関数の外で定義された変数
 - ➡ グローバル変数
 - ➡ 多用は要注意

変数のスコープ

```
In [1]: 1 #グローバル変数  
2 aGlobal = 100
```

```
In [2]: 1 def func1():  
2     x = aGlobal  
3     return x
```

```
In [3]: 1 def func2():  
2     aGlobal = 2 #このaGlobalはこの関数内のローカル変数
```

```
In [4]: 1 def func3():  
2     y = 1
```

```
In [5]: 1 print(func1())  
2 func2()  
3 print(aGlobal)  
4 print(y)
```

```
100  
100
```

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-5-3b7f52ebec91> in <module>  
      2 func2()  
      3 print(aGlobal)  
----> 4 print(y)
```

```
NameError: name 'y' is not defined
```

引数名を指定した呼び出し

- ➡ 引数名を指定することで、引数順序に従わずに利用できる

#引数名の指定

```
x1,x2=quadratic(c=1,b=2,a=1)  
print(x1,x2)
```

引数の省略

```
def squareSum(list,s=0):  
    for d in list:  
        s += d*d  
    return s
```

```
data = [4,2,6,4,1]  
s = squareSum(data)  
print(s)
```