

# 関数を作る (v.s. サブルーチン)

---

- 値を返す
  - return文を忘れずに
  - printだけでは人間には見えても使えない
- 関数の中ででプリントしない
  - デバッグ中ならともかく, 非常に使いにくい
  - それが本質的な機能である場合は除く
- 正しい答えを返す
- エラーの処理を行う

# テストプログラムを書く

---

- 同じファイルに普通に書く ×
  - モジュールとして読み込んだときに困る
- 同じファイルにガードを付けて書く △

```
if __name__ == "__main__":  
  
    print "start test"  
    print "fact1"  
    print "fact1(0)=",fact1(0)
```

- 別ファイルにテストプログラムを書く ○

# テストデータ

---

- 普通のデータ
  - 4, 5
- クリティカルデータ
  - -1, 0, 1, 100
- 予想外のデータ
  - 5.6, 'a'
  - 整数のサブクラス

# 非定義域引数の処理

---

- プログラムが止まらない ×,
- 何もしないでそれらしく値を返す ××
- 何もしないでエラーになる ○
- エラーとして処理する ×, △, ○, ◎
  - 続きはエラー処理で

# エラーの処理

---

- 普通ではない値を返すまたは値を返さない ○
  - 返ってくる値がどこかに明示されていること
  - エラーの戻り値はそのままでは使えないこと
- プリントする △
  - 人間には分かっても, それを使っている他のプログラムからは判断できない
- それなりの値が返ってくる ×
  - エラーはバグの兆候. 通り過ぎてはいけない
- 例外を発生する ◎
-

# 例外を発生させる.

---

```
def fact1(n) :  
    rslt = 1  
    if not isinstance(n,int) :  
        raise TypeError  
    elif n < 0 :  
        raise ValueError  
    else :  
        for i in range(2,n+1) :  
            rslt *= i  
        return rslt
```

# 補足：例外を捕まえる

---

```
import sys

def test(func, values) :
    for vv in values :
        try:
            aa = func(vv)
            print func.func_name+"("+vv.__str__()+")=",aa
        except :
            print func.func_name, "with argument", vv, ", \
                error raised:", sys.exc_info()[0]
```

- すべての例外をキャッチするのは邪悪とされている。
- `except` では処理できる例外を指定して捕まえるのが良い。