1 もくじ

2 関数

関数の定義

以下のように定義します.

```
def func_name(argv1, argv2, …):
    処理


    return # もし戻り値がなければ書かなくても良い.
```

C言語と大きく違うところは**戻りの型を書かなくて良い**ことと, **引数の型を指定しなくて良い** ところ.

関数の利用

実際に関数を使うプログラムを以下に書いてみます.

```
1 def func(x):
2         return 2*(x**2) + 3*x + 4
```

```
3
4
5  def main():
6          print("func(10) :", func(10))
7          print("func(2.4) :", func(2.4))
8          print("func(-3.4) :", func(-3.4))
9
10
11 if __name__ == '__main__':
12         main()
```

```
func(10) : 234
func(2.4) : 22.72
func(-3.4) : 16.919999999999998
```

再帰関数

関数Aの中で関数Aを呼ぶような, 自身の中で自身を呼ぶ構造を取る関数を `再帰関数` といいます.
Pythonでももちろん再帰関数は実装できます.

```
1  def decrement(x):
2          print(x, end="")
3          if x - 1 > 0:
4                  print(", ", end="")
5                  decrement(x - 1)
6          else:
7                  print()
8
9  if __name__ == '__main__':
10         decrement(10)
11         decrement(2.4)
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1
2.4, 1.4, 0.3999999999999999
```

関数の引数

位置引数とキーワード引数

Pythonでは, 関数を呼び出す際に, どの仮引数へ渡すかを選択することができます.

```
1  def func(a, b, c, d):
2          print(a*1e3 + b*1e2 + c*10 + d)
3
4  def main():
5          func(1, 2, 3, 4)
6          func(a=1, b=2, c=3, d=4)
7          func(c=1, a=2, d=3, b=4)
8          func(1, 2, d=3, c=4)
9
10 if __name__ == '__main__':
11         main()
```

```
1234.0
1234.0
2413.0
1243.0
```

関数の定義時に決めた仮引数の名前を `name` とすると, `name=val` のように, 渡す引数を選択することができ, これを**キーワード引数**といいます.
キーワード引数を用いずに, 指定された順に渡す引数を**位置引数**といいます.

位置引数とキーワード引数は**混在できます**が, 必ず位置引数を全て指定し終えてから, キーワード引数を使わなければなりません.

デフォルト引数

以下のプログラムを見てください.

```
def func(a, b, c="hoge", d="fuga"):
        print(a, b, c, d)

def main():
        func(1, 2, 3, 4)
        func(1, 2, 3)
        func(1, 2)
        func(1, 2, d=99)

if __name__ == '__main__':
        main()
```

```
1 2 3 4
1 2 3 fuga
1 2 hoge fuga
1 2 hoge 99
```

一部, 引数を与えていなくても動作している部分があると思います.
このように, 引数を受けるが, もし引数が与えられなくても, 予め指定しておいた値を引数としてうけるような引数を**デフォルト引数**といいます.

これは関数の定義時に設定することができます.
制約としては, デフォルト引数は, そうでない引数の右側になければならないという点です.
そのため以下のような実装はエラーがでます.

```
def func(a, b="hoge", c=10, d="hoge"):
        print(a, b, c, d)
```

可変長引数

今回は省略.
かなり便利なので興味があれば調べてみてください.

関数の戻り値

戻り値の設定

C言語のように戻り値の型を指定する必要はない.
なので, どのような型の値を返してもいいし, なんなら何も返さなくても良い.
戻り値は `return` で返せる.

```
def func(a, b, c, d):
        return a + b + c + d

def main():
        total = func(1, 2, 3, 4)
        print(total)

if __name__ == '__main__':
        main()
```

```
10
```

複数の戻り値を返すには

以下のように戻り値としてリストやタプルを返してあげればよい.
受け取る際はシーケンスをまるごと受け取ってもいいし, アンパックして複数の変数へ分割させても良い.

```
def func(a, b, c, d):
        return (a+b, c+d)

def main():
        a, b = func(1, 2, 3, 4)
        print(a, b)

if __name__ == '__main__':
        main()
```

```
3 7
```

3 スコープの話

どの言語においても, 変数の有効な領域というものがある.
ある関数内で定義した変数は, その関数を出ると有効ではなく成っていたり, forやwhile等の中で定義した変数もそれらからでると有効でなくなる.

このように変数の有効な範囲というのをスコープという.

一般的に, 変数のスコープというのは必要最小限であればあるほどよいとされる.

4 if __name__ == “__main__” の意味

__name__とはなにか

特殊な変数

`__name__` は特殊なグローバル変数で, スクリプトファイルが実行される際に, 自動的に値が設定されます.
(ちなみにこれ以外にも特殊なグローバル変数はいくつかあります.)

格納されているもの

`__name__` には, **コマンドラインから直接呼ばれたか, 他のスクリプトファイルから呼ばれたか**が格納されています.

- コマンドラインから直接よばれた時 -> `__name__ == "__main__"`
- 他のファイルからimportされた時 -> `__name__ == "呼び出し元ファイル名"`

なぜこれをするのか

理由は大まかに以下の二点に集約されます.

1. グローバルのスコープが汚れるため
2. モジュールとして利用するとき, 不要なプログラムが実行されるのを防ぐため

グローバルのスコープが汚れるという点に対して対応するため, 自分は `if __name__ == '__main__'` 内で `main()` を呼び, その中で主な処理を書くというステップを踏んでいます.

```
def main():
        主な処理


if __name__ == '__main__':
        main()
```

5 問題

数学的な関数を作る

作るもの

- 関数$f(x) = \alpha(x^2 - a) + b$をPython上で実現する
- 関数の引数は `x` , `a` , `b` , `α` の4つ
  - ここで, `a` , `b` , `α` はデフォルト引数とすること.
- 関数名はなんでもOK

プログラム例

```
1  def f(x, a=10, b=20, alpha=2):
2          return alpha*(x**2 - a) + b
3
4  def main():
5          print(f(10))
6          print(f(10, 10, 10, 1))
7          print(f(10, 5, 3))
8
9  if __name__ == '__main__':
10         main()
```

```
200
100
```

```
193
```

フィボナッチ数列 (深さ優先探索)

作るもの

- 再帰関数を利用してn番目のフィボナッチ数列を求めるプログラムを書く.

ちなみに, ここではフィボナッチ数列を$1, 1, 2, 3, 5, 8, 13, \cdots$のように定義します.

再帰関数は少し考え方が難しいので, 多めにポイントを記述していきます.
わからなければじゃんじゃん見てください.

ポイント1

- 問題を細かい問題に分解して考える.

ポイント2

- 再帰関数の中身を実装するときは, 再帰関数が完成しているものとして考えると良い.

ポイント3

- 再帰関数は大抵以下の構成で成り立っていると考えるとうまくいく(と思いたい).

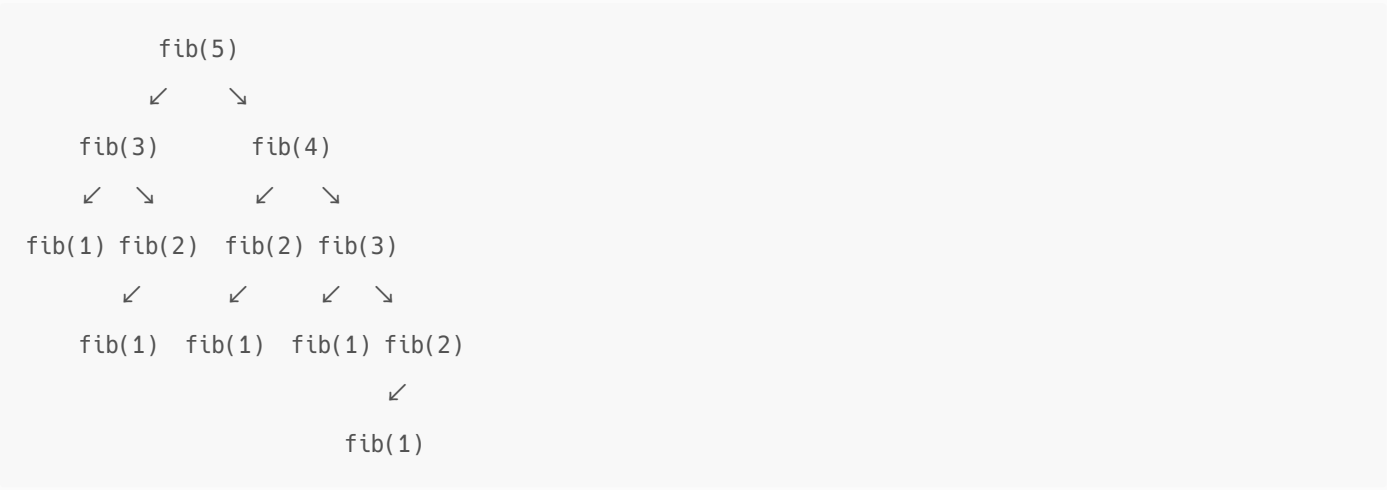
```
def recursive_func(arg1, arg2, …):
        if 再帰の末端に来たと判断するための条件:
                末端に来た時の処理


        return 自身の１つ前の段階から得る，自身の解
```

ポイント4

フィボナッチ数列の問題を分解すると以下のような2分木ができる.

(以下はn=5の時)



つまり, `fib(5)` では `fib(3) + fib(4)` を `return` すればよく, `fib(4)` では `fib(3) + fib(2)` を `return` すればよい.
これらより, `fib(n)` では `fib(n-1) + fib(n-2)` を `return` すれば良いことになる.

ポイント5

2分木を見れば, 終了条件(末端条件)は `n=1` であることが挙げれられ, そのとき, フィボナッチ数列の1番目を返せば良いので, `1` を返せば良い.
しかし, これだと `fib(2)` のときに `fib(2 - 1) + fib(2 - 2)` でエラーになりかねないので, `n=2` のときも条件に加え, フィボナッチ数列の2番目も1なので, こちらも同様に `1` を返せば良い.

プログラム例

以下に自分なりのプログラムを記述します.

```
1  def fib(try_num):
2          if try_num <= 2:
3                  return 1
4          return fib(try_num-1) + fib(try_num-2)
5
6  def main():
7          print(fib(9))
8          print(fib(8))
9          print(fib(7))
10         print(fib(6))
11         print(fib(5))
12         print(fib(4))
```

```
        print(fib(3))
        print(fib(2))
        print(fib(1))

if __name__ == '__main__':
        main()
```

```
34
21
13
8
5
3
2
1
1
```