難易度：　★★★☆☆

押忍！（私も例にもれず押忍で始めます w）

はじめまして。株式会社セガの岸本と申します。Softimage に関してコラムを書かせていただくことになりました。今後ともよろしくお願いします。

Softimage × Python

Python には魅力的な機能がたくさんあります。でも実際 Softimage 上でそのすばらしい機能を使ってみようと思ったとき、「あれ？これ Softimage ではどうやって使えばいいんだろう？」と悩むことがしばしばありました。情報をなかなか見つけられずに苦労したのを覚えています。

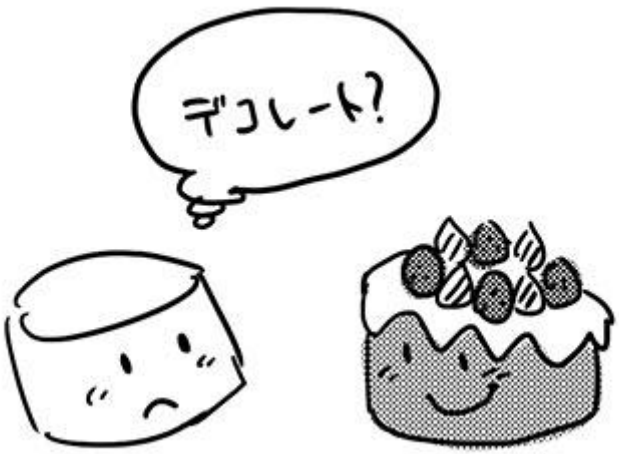
そんなわけで、私のコラムでは Softimage × Python をうまく使うための Tips を紹介していければと思います。

デコレータを使ってみよう！

第一回となる今回は VBScript、JScript にない、Python ならではということを意識して「デコレータ」を紹介したいと思います。

デコレータとはその名のとおり関数をデコレートするための機能です。

ここでいうデコレートってどういうことでしょう？



例えば

- スクリプトの処理が重い！どこが重いか知りたいからそれぞれのコマンド実行時間を計測できるようにしたい！
- コマンドを実行した後元に戻すのにアンドゥを連打しなきゃいけない！一回で戻るようにしたい！
- SI のバージョンによってはうまく動作しないコマンドがある！特定のバージョンでは何もしないようにしたい！

などなど、「ちょっとだけコマンドに機能を追加したい！」と思ったことはありませんか？

この機能追加が「デコレート」にあたります。

これらは関数内部に目的のコードを書くことで実現できるものばかりですが素直に書いてしまうとちょっと困ることになります。

関数の実行時間の計測するスクリプトを書くとしたらこんな感じでしょうか？

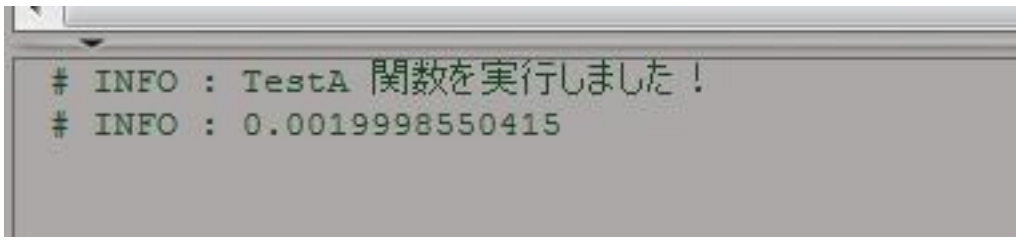
Softimage のスクリプトエディタにペーストして実行してみてください。

```
import time #  時間に関する機能を持った time モジュールを読み込む


# テスト関数の定義
def TestA():
    starttime = time.time() #  処理実行前の時刻をとってくる
    # この関数でやらせたい処理 ----------------------↓
    LogMessage(u'TestA 関数を実行しました！')
    # ---------------------------------↑
    endtime = time.time()       #  処理実行後の時刻をとってくる
    LogMessage(endtime - starttime) #  実行後の時刻から実行前の時刻を引いたものを表示


# テスト関数の実行
TestA()
```

私の環境(Softiamge2013)ではヒストリペインにこのようなログが出ました。



テスト関数「TestA」の実行結果に続いて経過時間が表示されます。単位は秒ですので、0.002 秒以下しかかかっていないことがわかります。「TestA 関数を実行しました！」とログ表示するだけなので早いですね。

ところでこのコード、ちょっと見づらくないですか？

関数自体が行いたい処理「print u'TestA 関数を実行しました！'」部分と経過時間を表示する部分がゴチャまぜです。それに実行時間をログするのにも処理時間がかかるのでログ表示したくないときがあるかもしれません。その場合最初と最後の２箇所をコメントアウトしないといけません。

なんとかもうちょっとすっきりしないものでしょうか？

デコレータはそんな要望にスマートに答えてくれます。
デコレータを使ったコードはこんな感じになります。

スクリプトエディタにペーストして実行するとさっきのスクリプトと同じような結果になると思います。

```
import time #  時間に関する機能を持った  time  モジュールを読み込む

# 時間経過のログを表示するデコレータの定義
def LogTime(func):

    # 与えられた関数の代わりに実行する新しい関数の定義
    def new_func(*args, **kwargs):
        starttime = time.time()        #  処理実行前の時刻をとっておく
        rtn = func(*args, **kwargs) #  与えられた関数(ここでは TestA)を実行して戻り値をとっておく
        endtime = time.time()          #  処理実行後の時刻をとっておく
        LogMessage(endtime - starttime)    #  実行後の時刻から実行前の時刻を引いたものを表示
        return rtn                          #  とっておいた戻り値を戻す

    # 新しく作った関数を戻す
    return new_func


# テスト関数の定義
@LogTime            # @ +  デコレータ名で直後に置かれた関数をデコレート
def TestA():
```

```
        LogMessage(u'TestA 関数を実行しました！')


# テスト関数の実行
TestA()
```

どうでしょうか？コードが長くなって複雑になったように思われるかもしません。

でも TestA 関数の部分を見てください。余計なコードがなくなってすっきりしたと思いませんか？これなら関数で本当に実行したい処理に集中してコードが書けそうです。

良く見ると今までなかった「@***」という行が追加されています。これがデコレータの呼び出し合図です。この「@***」を関数の前につけると、その関数は「***」デコレータによってデコレートされることになります。どのようにデコレートするかは上の LogTime 関数に書かれています。

わかりにくいデコレータ

見ていただいたようにデコレータを使えば1行追加するだけで簡単に機能追加できてとても便利です。

ただこのデコレータ、便利なんだけどちょっとわかり辛い。それは見えないところでデコレータ用の特別な処理が行われるからです。

少しでもわかりやすくするためにどんな処理が行われているのかマンガにしてみました。

よけいわかり辛いでしょうか ｗ？？？

厳密にはツッコミどころ満載だと思いますがきっと裏でパイソンさんはこんな感じのことをしてるんじゃないかと思います。

デコレータは対象となる関数が作られるタイミングでその関数を受け取って実行され、Python はデコレータから戻ってきたものを元の関数と置き換えます。

今回の例では対象となる関数(TestA)が引数 func としてデコレータに渡され、内部でその func を実行して経過時間を表示するようにつくられた new_func 関数を作成し戻しています。元の関数(TestA)は見えないところでこの新しい関数に置き換えられているので TestA を実行しようとしたとき実際には新しい関数(new_func)が実行されて経過時間が表示されるという訳です。

おおざっぱに言うとデコレータは「関数を置き換える特別な関数」、ということができると思います。

まとめ

なんとなくデコレータの便利さと、どういうものなのかわかっていただけたでしょうか？

デコレータには以下のようなメリットがあると思います。

- 対象関数の内容とは完全に切り離すことができる
- 開始と終了に処理を追加しなければならない場合も 1 行で済む

関数の内容と直接関係ないようなコードをたくさん書いているような場合にはどんどんデコレートしちゃいましょう。

ここまで来て「あれ？」と思った方もいらっしゃるかもしれません。このままだと、「Softimage ではどう使うのか？」という冒頭でお話した部分がすっぽり抜けてしまっています。

今回のコードはスクリプトエディタで実行しないと動きません。せっかく作ったデコレータはほかのいろんなスクリプトから呼び出して使いたいですよね？デコレータは Softimage のコマンドに対しても使えますが、デコレータ用につくった関数はコマンドにできません。コマンド化すると他からの呼び出しはできますがちゃんとデコレータとしては動いてくれなくなってしまいます。

デコレータをどこからでも呼べるようにするには Python 自体にカスタムモジュールとして登録する必要があります。そしてそれは Python を Python らしく使うために欠かせない部分だと言っても過言ではありません。

ということで次回は簡単にカスタムモジュールを登録する方法についてご紹介し、今回作ったデコレータをどこからでも呼べる状態にしたいと思います。