基本

定義：

```
function func_name(param1="default_val", param2="default_val"...) {
    ....
}
```

驚くべきは、呼び出し。

- 引数指定に括弧をつけてはならない。
- 引数はスペースで区切る。

厳密には、引数が 1 つしか無い場合には括弧を付けても変わらないのだが、(p1 p2 p3...) としてしまうと、PowerShell には「括弧付きでスペースで区切られた値のコレクション」という記法が存在しないので、これはエラーになる。

```
func_name p1 p2 p3 ...
```

と呼び出す。面妖な。。。
もし、

```
func_name(p1, p2, p3...)
```

のように呼び出すと、これは引数として配列 (p1, p2, p3...) を 1 つ渡した、という意味になってしまうので注意が必要。

戻り値

関数内のリテラル値、変数に代入されていない文や式の関数内実行結果戻り値、return 文の引数がすべて、戻り値となる。
戻り値が複数ある場合、すべてが含まれた 1 つのオブジェクト配列として呼び出し元に戻される。

こういうだらしない仕様のため、return 文の意味があまりない。関数を中断したい場合にしか使用できない。
戻り値はリテラル値でもいいので、関数定義内にそのまま置いておけばよい。

```
function aaa {
    ...
    ...
    $returned_valu
}
```

return 文は必要ない。

また、関数内で実行された文や式の戻り値で変数に代入されなかったものもすべて、戻り値に含

まてしまう。
戻り値となる値が複数あると、それらが含まれる配列が戻される、という直感的ではない仕様のため、

```
function bbb {
    ...
    $args[0] -match ".*regex (pattern).*"
    if ($Matches -ne $null) {
        ...
    }
    ...
    $returned_value
}
```

のような関数があると、$args[0] -match ".*regex (pattern).*" はブール値を返すので戻り値配列に含まれてしまい、意図した動きをしなくなってしまう。プログラマは $returned_value だけを関数の戻り値として受け取りたいはずだ。

これはバグを誘発する非常に悪い仕様だと思う。
プログラマは、関数内で実行される文や式で、戻り値を返すものをすべて把握しなければならない。
戻り値を捨てるためには、

```
[void] ($args[0] -match ".*regex (pattern).*")
```

または

```
$null = $args[0] -match ".*regex (pattern).*"
```

とする。