目次へ

変数の有効範囲

変数の有効範囲のこととか説明しておいたほうがいいですよね、多分。

例をあげると、下のようなスクリプトがどういう動きをするか、とかそういうやつです。

```
def ojama():
  a = 0
  return

a = 10
ojama()
print a
```

ojama という関数が a の中身をゼロにしてしまうから、この場合最後の print a は 0 を表示するんじゃないかな、と考えてしまうかもしれません。でも、試してみると分かりますが、これの結果は 10 が表示されます。

なんでだろ、というあたりを、変数の有効範囲を知るためのきっかけとしましょう。

変数の有効範囲のことを「スコープ」と呼ぶときもあります。なるほど、スコープをのぞいて見える範囲が「有効範囲」ってことですかねえ。

まず、関数の中で使っている変数は関数の中だけで完結するという原則があります。関数の実行が return とかで終了しちゃったら、その関数で使われた変数は消えちゃうということです。最初の例なら、ojama という関数の中で使った a は、その下の命令で使おうとした a と全然別物ということになります。これは便利なこととです。関数が動作することで外界に余計な影響を及ぼさないで済みますからねえ。とても長いスクリプトを書いていて、あとになって関数をひとつ

書き足すと便利になるなと思ったと考えてください。「有効範囲」というのが関数の中だけに限られないとしたら、作業のために一時的な変数をいくつか用意したくても、関数の外でだれかがすでに使っているようなものだったらどうしよう、とか悩むことになりかねません。

で、逆の場合はどうかという点を次に見ます。下の例。

```
def dump():
  print a

a = 30
dump()
```

このとき、dump 関数の中で a の中身（これも外界で定義されたやつですね）を表示すること、これはOKです。結果として、30 という表示がされるでしょう。これならいいわけ？

はい、このときはいいのです。dump関数の実行中に 'print a' という命令を見たときの python の気持ちはこんなふうでしょう。

「a という変数を print したいのか。今実行している dump 関数の中では、特に a を定義している場所はないな。じゃあ、それより元はどうだろう。関数の呼び出し元には、a があるようだ。中身は 30 とある。じゃあこいつを使うことにしよう」

ってことで、30 という結果が得られるんです。dump の内部では a に何も入れてないところがミソです。こうすれば、次の探索範囲として、関数を呼び出した元にある変数が探されます。（それでも見つからなかったら、さすがにエラーを起こしますよ）

関数の引数につかう変数（仮引数と呼ぶこともあります）は、関数の内部で定義した変数と同じ扱いです。下の例。

```
def doublenum(su):
  return su * 2

su = 10000
print doublenum(7)
print su
```

これの結果はふたつ表示されて、最初が 14 、次が 10000 です。関数の外で su という変数に10000を入れたのは、double 関数を実行したあとでも壊れないで残っていることがわかります。

「関数の外で定義した変数」とさっきから何度か書きましたが、どうも長ったらしい言い方でアレです。もうちょっと技術屋さんっぽい言い方として「グローバル変数」と呼ぶこともありますので、一応覚えておくとよいでしょうか。このときは、関数の中にあるやつは「ローカル」変数です。グローバルって、妙に壮大な言葉を持ち出してくるもんだ。一般にそう呼ばれているんだから、筆者もこの用語を使いますけどね。プログラマーどうしが「グローバルが…」「あれもグローバルで…」とかそんな会話をしているのを立ち聞いたとしても、別に世界を相手になにかをしているわけではないので安心？してください。

このくらいのことを納得してくれていれば、今後そうそう困ることはないと思います。変数の中身が思ったものと違うみたいだけど？　という事態になったときに、ああ、変数のスコープから外れちゃったからか、とか思い当ってくれればよいです。

わかる人へ：今回は global 文のことは説明しません。