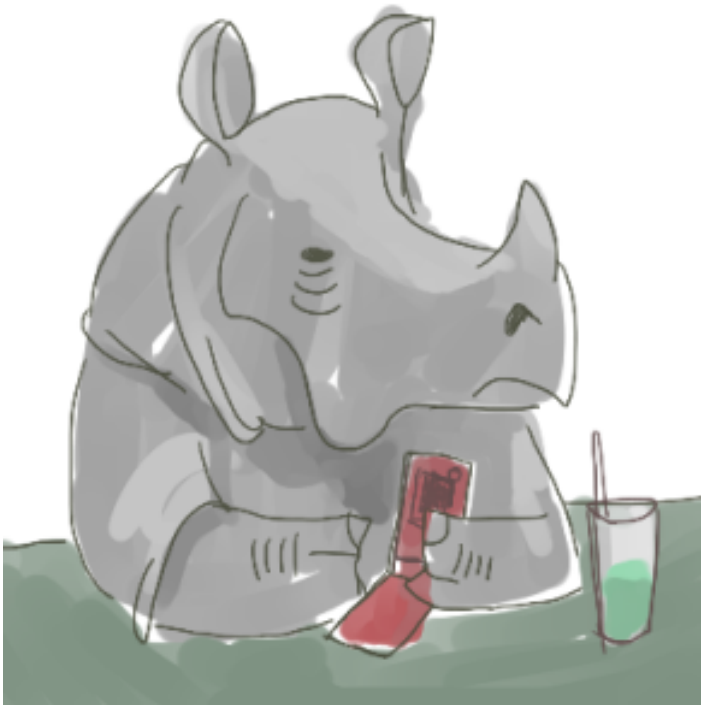
目次へ

文字列



文字列(1)

前章までは、数字ばっかりを扱いました。ここからは文字も扱えるようになります。

まず、"文字列"という用語が、初めての人には目新しいんじゃないでしょうか。プログラム屋さん達にとっては、あまりに当たり前の用語になっていますが。

文字が集まって並んだのが文字列です。だいたいそんな理解で間違いありません。次の例を試してみましょう。「'」という記号は、クォーテーションと呼びます。打ち込むのは、普通の日本語キーボードなら、シフトキーを押しながら[7]です。

```
>>> print 'Hello World!'
Hello World!
```

この、Hello World! というのが、ひとつの文字列です。pythonでは、クォーテーションで囲まれているものが文字列です。これで、数字じゃなくて、メッセージみたいなのも表示できるようになりますね。

文字列も、変数に入れることができます。下のような感じ。

```
>>> a = 'Hello World!'
>>> a
'Hello World!'
```

数以外のものも、変数にすんなりと入ります。見た目は、数のときと同じくa のまんま。中身は、確かめてみるまでわかりません。

今回の表示が、クォーテーションごと表示されたのに気づいたでしょうか。print を使ったときは、クォーテーションが外れて表示されているのに。この違いを説明するのはあまり簡単ではないので、さしあたりは気にしないでください。printを使うときと変数を直接確認するときでは、すこし表示に差が出ることがある、という程度に理解してもよいです。どちらかといえば、print を使うほうが、より"本式"な表示となります。なんだかあいまいですみませんが。

ところで、aには数が入ったり文字列が入ったりできますね。aに文字列が入っていたときは、足し算なんかできるのでしょうか。

```
>>> a + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

怒られました。文字列と数の足し算なんかできないよ、といってます。pythonでプログラムを作るときは、変数にどんな種類のものが入る予定かをあらかじめちゃんと考えておかなくてはいけません。今は変数に数か文字列が入るだけですが、今後、かなり色々なものが変数に入るようになります。中身が何かによって、それをどう操作できるかがぜんぜん違ってきます。なんでも入る分、これはこれで厄介な性質でもありますので、注意してくださいね。（後々、この厄介さが、pythonのパワフルな特徴に「化ける」ことになります。そこまでうまく説明を持っていけるかどうかは筆者次第ですが...）

ところで、文字列の中に「'」クォーテーションを含めることはできるで しょうか。たとえばこんな感じ。

```
>>> a = 'I'm a student'
  File "<stdin>", line 1
    a = 'I'm a student'
          ^
SyntaxError: invalid syntax
```

怒られました。I のあたりで一旦文字列として完成してしまいますから、残りはじゃあ何なんだよ、という怒られ方です。

こういうときは、クォーテーションの代わりにダブルクォーテーションを使って文字列を囲んでもよいです。「"」記号の出し方は、シフトキーを押しながら[2]。

```
>>> a = "I'm a student"
```

これなら紛らわしくないですね。中身に従って、どっちの記号で文字列を表現することにしても全く同じです。普段はどっちの記号を使うかを決めておいて、不都合が起こったときはもうひとつの記号をつかってしのぐ、という方針でやっていけばよいでしょう。参考までに筆者は、どっちを使うかがひとつのスクリプトの中でも気分次第でバラバラです。すんません。

当然ここで発生する疑問について、ひとつ。「ひとつの文字列の中に、" も ' も両方あるときは、どう表現すればいいんじゃ」ごもっとも。これだといよいよ小手先の対処では済みません。こう表現してください。

```
>>> a = "I'm reading ¥"The Hobbit¥"."
>>> print a
I'm reading "The Hobbit".
```

途中、「ここの記号は文字列の中身だから、勘違いすんな」ということを明示的に示すために、「\」という記号を一文字前にくっつけるという方法がありますので、これを採用してみました。これだと、ダブルクォーテーションをダブルクォーテーションで囲むという表現が可能になります。

（キーボードから、この「/」の逆に傾いた記号（バックスラッシュっていいます）が打ち込めないよ、というかたは、代わりに¥マークを入力してください。全く同じことが起こります。）

ただのクォーテーションで囲む場合は、下のような感じの表現になりますね。

```
>>> a = 'I¥'m reading "The Hobbit".'
```

I'm の中のクォーテーション記号が勘違いを招くので、「\」をつけて意図を明らかにしました。結果は、先の例と全く同じことです。

文字に「\」をつけて特殊な意図を示すことを、「エスケープする」と言うときがあります。このテキストでも、今後時々この用語を使うことがあるかもしれません。余計な勘違いから「逃げている」という意味に捉えておくと、覚えやすいでしょうか。

もうひとつ、意地悪な質問があってもよさそうですね。「\自体を扱いたいときはどうすんのさ」こうします。

```
>>> print "Price is ¥¥100."
Price is ¥100.
```

ここで、\をひとつだけにして試すと、変な表示になります。Price is @. とかね。理由はここではまだ詳しく説明しません。\は思った以上に高機能な記号なんです。完全に使いこなすのは、なかなか。さしあたりは、クォーテーションの「エスケープ」に使う場合と、\100の表示をする場合だけを覚えましょう。（¥マークが表示される環境を想定しています。今の例は、つまり100円って表示させようとしているわけで。）

そうだ、さっきは文字列と数を足し算しようとして失敗しましたが、実は文字列と文字列を足し算することはできます。

```
>>> "Shidoro" + "Modoro"
'ShidoroModoro'
```

足し算というより、文字列がつながった形になりましたね。pythonにとっては、これが「文字列の足し算」です。

一見数字に見えるようなものでも、pythonが文字列だと認識している限りはまともな足し算にはなりません。

```
>>> "1" + "2"
'12'
```

ダブルクォーテーションで囲んだからには、人間にとっては数に見えてもpythonにとっては文字列に過ぎません。足し算させても、こんなトンチみたいな答えが返ってきます。たまにハマるところですので、注意しておきましょう。

さて、文字列どうしの引き算はできるでしょうか。

```
>>> 'abc' - 'c'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

pythonは、文字列から文字列を「引き算」する方法は持ちあわせないようです。今の例だと、あるいは 'ab' なんて答えが返ってこないかな、と期待してしまいますが。

ちょっと変わった例ですが、文字列と数（整数）を「掛け算」することはできます。何が起こるでしょうか。やるまえに予想してみるのもよいでしょう。

```
>>> "Hello" * 10
'HelloHelloHelloHelloHelloHelloHelloHelloHelloHello'
>>> 10 * "Hello"
'HelloHelloHelloHelloHelloHelloHelloHelloHelloHello' （これでも同じ）
```

まあ、そういうことです。こんなの使い道あるのかよ、と言われそうですが、なくもないですよ。文字だけで罫線を引きたいときに、ーーー といった感じの文字列を直接スクリプト内に書くんじゃなくて、"-" * 80 とか書いたほうが短く済むし、ー- を ====== に直したくなった場合も、一文字直すだけでいいじゃない。

文字と実数（10.5みたいなの）を掛け算することはできません。意味がわからんですからね。やって、怒られて、確かめておいてください。

ところでここまで、日本語の文字列を扱うことを意図的に避けています。もちろん、こんな書き方をして、普通に日本語とか他の言語を扱うことができるんですよ。

```
>>> print "こんにちは世界!"
```

でも、Windowsのコマンドプロンプトで日本語を打ち込むのは少し難しいですし、ちょっと説明の難しい制限事項もあるもので、しばらくは英字だけで我慢しておいてください。もちろん、こっそり日本語の文字列なんかも試して結構ですが、予想しない表示とかが出てきても、今のところはあきらめておいてくださいね。スクリプトファイルを書いてこれを実行できるようになれば、そういう制限事項は一切なくなります。あくまで対話シェルを使っているときだけの問題ですので、 そこらへんはご安心ください。

ああもうひとつ説明すべきものがありました。空文字列です。

```
>>> a = ""
```

長さがゼロの文字列です。これも、pythonにとっては立派な文字列です。使い道はちゃんとありますが、今はこれが文字列だと覚えておくだけでよいです。

文字列(2)

ちょっとした仕事で、文字列をいろいろと操作することは多いものです。名簿を扱う、メールアドレスやURLを扱う、電話番号を扱う、書誌を扱う、云々。

pythonは、文字列にちょっとした操作を行うのがとても快適にできています。下に順に挙げていくようなものを試しながら、それらの使い道について色々と想像をめぐらせてみましょう。

まずは、文字列の長さを調べる方法。

```
>>> len("internationalization")
20
```

len( )という書き方で、文字列の長さを取得できます。internationalizationは、20文字の単語のようですね。文字列を変数に入れている場合でも、同じようにできます。

```
>>> a = "internationalization"
>>> len(a)
20
```

次は、文字列のなかから好きな部分の一文字を取り出す方法。変数aに、aからzまでの文字を全部並べたものを入れることにします。打ち込むのが面倒ですが、この例で今後いろいろやるので、入力しといてください。

```
>>> a = "abcdefghijklmnopqrstuvwxyz"
>>> a[10]
'k'
```

aからzまで並んだ文字列の10番目を、こんな[ ]の中に位置を入れて指示することで取り出すことができます。丸カッコ( )じゃだめですよ。あれ、アルファベットの10番目は j だろう、と考えてしまいそうですが、実はpythonでは左端を0として数え始めます。だから[10]ってのは実質11番目のことなんです。これは慣れが必要です。

[10]といった感じに指定するのではなく、この中身にも変数を使ってよいです。もちろん、変数には文字列でなくマトモな数を入れておかなくちゃだめですが。

```
>>> i = 20   ←iっていう変数を使おう
>>> a[i]     ←さっきの a をそのまま流用しています
'u'

>>> i = "7"   ←文字列を位置として指定したら？
>>> a[i]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: string indices must be integers, not str  ←怒られる
```

アルファベットは26文字なので、[25]を超える位置を指定すると怒られます。下の例のとおり。

```
>>> a[100]        ←100番目の文字はなに？
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range   ←範囲外の指定はダメ
```

ちょっと変わった位置指定。負の数を指定したら？

```
>>> a[-1]
'z'     ←ほほう…
```

マイナス1番目、という指定は、文字列の最後の一文字を取り出すことを意味します。同様に、[-2]としたら 'y' が取り出されます。文字列の長さはどうでもいいから、最後付近の文字が知りたいんだよ、というときに使えます。文字列の最後から数えるときは[-0]から始まらないんだね、と考えた人は いますか。-0は0と同じですから、これだと自動的に"最初の一文字"ってことに なってしまうんです。

これで、文字列の好きな場所から一文字持ってくるということについて、大体の説明をしました。次は、好きな場所から、好きな長さ分だけ持ってくることについて。（以後、こういうものを、"部分文字列"と呼ぶことがあります）

ちょっと指定のための表記が変わります。ちょっとだけです。

```
>>> a[10:20]
'klmnopqrst'
```

「:」のことを、今後コロンと呼びます。始まる位置と終わる位置をコロンでつなげるという書き方をすると、「どっから、どこまで」と指定したことになります。このやりかたを「スライス」と呼ぶこともあるんですが、覚えることばかりで面倒ならあまり呼び方にこだわることはありません。

注意深い人は、a[10]は確かに 'k' だけど、a[20]は 'u' じゃないか、't'で終わってるのでは一文字足りないぞ、と気づいているでしょう。こういうルールなんだよ、とだけ言ってしまうこともできますが、もうちょっと納得しやすい説明も可能です。[10:20]のような指定をするときは、それぞれの数字は「文字どうしのスキマ」の位置を指定していると想像してください。

```
   0   1   2   3   4   5   6
   | a | b | c | d | e | f | g  ....
```

こう想像しながら考えれば、たとえば[1:5]というのは、aとbの間にザックリ刃を入れて、次にeとfの間にも刃を入れて、切れたものを持ち上げる、というイメージになるでしょう。このときできあがる部分文字列は、bcde ですね。これなら無理はないと思うのですが如何。

この[何:何]という指定は、他にも何種類かのバリエーションがあります。

負の数での指定を含む

このときも、例えば-1という指定は「最後の切れ目」と読めます。下の図示のとおりです。-1番目の切れ目の位置に注意。

```
  -5  -4  -3  -2  -1
   | v | w | x | y | z | 文字列オシマイ
```

```
>>> a[-5:-2]
'vwx'
```

終わりが指定されていない

このときは、指定の切れ目から文字列の最後まで、という意味です。

```
>>> a[20:]　←終わりの位置を省略している
'uvwxyz'
```

始まりが指定されていない

こっちは逆で、最初から指定の切れ目まで、です。

```
>>> a[:5]　←開始位置を省略している
'abcde'
```

どっちも指定していない

？？？

```
>>> a[:]　←開始も終了も省略している。なんだこりゃ？
'abcdefghijklmnopqrstuvwxyz'
```

最後の例はなんの冗談だよ、という感じですねえ。でも、あとになって意外な使い道があることが分かってくる、かも知れません。そこまで詳しい説明に到達できるかは分かりませんが。

実際に使うとすると、こんなふう

文字列の一部分を取り出すことについて、さしあたりこんなところでしょうか。

これらと同時に説明しておきたいのは、文字列の中を検索する機能です。少し目新しい書き方ですが、まずマネしてみてください。

```
>>> a.find("f")　← f っていう文字は何番目にありますか
5　←5番目だよ（ゼロから始まるので、実質６番目。以下このテの注記は省略します）
```

aという文字列（または文字列の入った変数）にピリオド記号「.」をつけて、そのあとでfind( )とつなげます。検索したいものは、一文字だけでなくても構いません。

```
>>> a.find("opq")　← opq って続く文字は、何番目から始まりますか
14　←14番目からだよ
```

```
>>> a.find("hello")　←hello って続く部分は見つかりますか
-1　←ねえよ
```

findを使って部分文字列が見つからなかったときは、-1が返ってきます。さっきみたいに、「最後から何番目...」という意味ではなく、単なる検索失敗ということです。

findの中身に指定する文字列も、もちろん変数を使って指定してよいです。例は示しませんが。

ここまで説明したところで、ひとつ応用問題です。。

ある文字列が、"Language:English" のように、「項目名:その値」といった具合にコロンでつながったものだとします。このときに、項目名と値をそれぞれ別に表示してみましょう。対話シェルの上で、何度も命令を打ち込んで構いません。

やってみましょう。検索結果などを新しく変数に入れたりしながら、一歩ずつ進めていきます。

```
>>> b = "Language:English"  ←まずは、変数に例の文字列をいれて、繰り返して入力
                              しないで済むようにしよう
>>> i = b.find(":")       ←コロンは何番目にありますか。いきなり表示しないで、
                             i という変数にそれを入れておいてね
>>> i           ←どんなのが入ったか確認
8               ←8か…
                  findってのは、文字列じゃなくて数を返してくれるようだ。当然か
>>> b[:i]       ←例の文字列の、最初から8番目（の切れ目）までを表示
'Language'
>>> b[i:]      ←例の文字列の、8番目（の切れ目）から最後までを表示
':English'      ←これではコロン記号まで混じってしまう
>>> b[i+1:]    ←じゃあ、そこからひとつ足した位置（の切れ目）から最後まで
                  表示させることに変更
'English'      ←期待どおりに出たよ
```

どうだったでしょうか。これっぽっちの仕事なのにずいぶん面倒くさいじゃないか、と思ったりしますか。確かにそうですね。でも、ここまでやって確立させた手続きを簡単に再現できるように仕立てておくことが、そのうちできるようになっていきます。"関数"というものの説明まで、それはもうちょっと待ってくださいね。

あと、この[ ]とか[:]とかで囲む指定方法についてちょっとしつこく説明したのは、あとでリストを扱うようになったときに同じ方法で扱うことができるからです。けっこう重要な基本知識なんです。