b) 34.7 を切り捨てによって整数に変換して下さい。
c) 34.7 を四捨五入によって整数に変換して下さい。
d) -86 の絶対値を取ってから、それを浮動小数点数に変換して下さい。

3章 文字列

数値はコンピューティングの基礎です。そもそも、コンピュータが発明されたのは、数値を操作するためだったくらいです。しかし、この世界には、住所、画像、音楽など、数値以外のデータが無数にあります。これらのデータはそれぞれ独自の型として表現でき、それらの型の操作方法を知ることは、プログラミングの能力の大きな部分を占めます。この章では、数値以外のデータの１つとして、この文に含まれる単語や DNA のらせん構造を表現する ATGC の４字の並びなどのテキストを表現する型を紹介します。説明の過程で、プログラムをもう少し対話的にする方法も考えていきます。

3.1 文字列

コンピュータはもともと算術計算のために発明されたものかもしれませんが、今日のコンピュータが大半の時間を費やしているのは、テキストの処理です。デスクトップのチャットプログラムから Google に至るまで、コンピュータはテキストを作成、保存、検索し、こちらのテキストをあちらに動かすということを繰り返しています。

Python では、テキストの塊は**文字列** (string)、すなわち**文字** (character。英字 =letter、数字、記号を含む) の並び (シーケンス) として表現されます。文字シーケンスを格納するためのもっとも単純な型は、str です。この型は、北米のほとんどのキーボードで入力できるローマ字を格納できます。それに対し、unicode というもう 1 つの型を使うと、漢字や化学記号、クリンゴン語に至るまで、ありとあらゆる文字を並べた文字列を格納できます。この章のサンプルでは、簡単な方の型である str を使っていきます。

Python では、値をシングルクォートかダブルクォートで囲むことによって、その値が文字列だということを示します。

strings/string.cmd

```
>>> 'Aristotle'
'Aristotle'
>>> "Isaac Newton"
'Isaac Newton'
```

始めと終わりのクォートは同じものでなければなりません。

strings/mismatched_quotes.cmd

```
>>> 'Charles Darwin"
  File "<stdin>" , line 1
    'Charles Darwin"
                    ^
SyntaxError: EOL while scanning single-quoted string
```

文字列を隣り合わせに並べれば、2つの文字列を連結できます。

strings/concat.cmd

```
>>> 'Albert' 'Einstein'
'AlbertEinstein'
```

Albert と Einstein の2語がすき間なく並べられていることに注意して下さい。単語の間にスペースを入れたければ、Albert の末尾か Einstein の先頭にスペースを追加します。

strings/concat_space.cmd

```
>>> 'Albert ' 'Einstein'
'Albert Einstein'
>>> 'Albert' ' Einstein'
'Albert Einstein'
```

しかし、文字列の連結には、+ を使った方がはっきりするでしょう。2個の文字列の被演算子を持つ場合の + は、**連結演算子** (concatenation operator) になります。

strings/concat2.cmd

```
>>> 'Albert' + ' Einstein'
'Albert Einstein'
```

+ 演算子は、数値の加算にも文字列の連結にも使われますが、このようなものを**多重定義された演算子** (overloaded operator) と呼びます。多重定義された演算子は、被演算子の型に基づいて異なる機能を持つのです。

もっとも短い文字列は、文字が0個の**空文字列** (empty string) です。次の例からもわかるように、空文字列はテキスト版の0と言うことができます。他の文字列に空文字列を加えても(連結しても)、何も変わりません。

strings/empty_string.cmd

```
>>> ''
''
>>> "Alan Turing" + ''
'Alan Turing'
>>> "" + 'Grace Hopper'
'Grace Hopper'
```


ここで1つ面白いことを考えてみましょう。+ 演算子は、被演算子として文字列と数値を1つずつ取ることはできるのでしょうか。もし可能なら、加算と連結のどちらの機能が使われるのでしょうか。実際に試してみましょう。

strings/concat3.cmd

```
>>> 'NH' + 3
Traceback (most recent call last):
  File "<stdin>" , line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
>>> 9 + ' planets'
Traceback (most recent call last):
  File "<stdin>" , line 1, in ?
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Python が TypeError を報告してきたのは、これで2度目です。1度目は「2.6.1 ローカル変数」のコードで、そのときの問題は関数に正しい数の引数を渡していないというものでした。ここでは、Python は、異なるデータ型の値を + しようとしたことに文句を言っています。それでは、数値の加算と文字列の連結のどのバージョンの + を使ったらよいかわからないというのです。

この場合、人間なら正解が何かはすぐにわかります。しかし、次の例はどうでしょうか。

strings/concat4.cmd

```
>>> '123' + 4
```

Python は、'1234' という文字列と 127 という整数のどちらを返すべきでしょうか。どちらもすべきではないのです。人間が求めていることを推測しようとすると、少なくとも何度かは Python は勘違いをします。そのようなとき、私たちは問題点を突き止めなければならなくなりますが、Python が勝手な推測をしていると、ヒントとなるエラーメッセージを見ることができなくなってしまいます†。

文字列の中に数字を入れたい場合には、組み込みの str 関数で文字列に変換してから連結をすれば簡単です。

strings/concat5.cmd

```
>>> '12' + str(34) + '56'
'123456'
```

† まだ納得できないようなら、'7'+0 は文字列の '70' になるのに対し、'7'-0 は 7 になる JavaScript (Web プログラミングで使われる言語) の動作についてどう思うか考えてみて下さい。

Python は、確かに + 記号では文字列と数値を結合できませんが、他の演算子でも一般的に文字列と整数を結合できないというわけではありません。たとえば、次のようにすれば、* 演算子を使って文字列の繰り返しを作ることができます。

strings/repeat.cmd

```
>>> 'AT' * 5
'ATATATATAT'
>>> 4 * '-'
'----'
```

整数が 0 以下なら、この演算子は空文字列（文字のない文字列）を返します。

strings/repeat2.cmd

```
>>> 'GC' * 0
''
>>> 'TATATATA' * -3
''
```

3.2 エスケープ文字

文字列の中にシングルクォートを入れたいものとします。次のように直接書き込むと、Python が文句を言います。

strings/single_in_single.cmd

```
>>> 'that' s not going to work'
  File "<stdin>" , line 1
    'that's not going to work'
          ^
SyntaxError: invalid syntax
```

何が問題なのかと言うと、Python が 2 個目のクォート（あなたが文字列の一部のつもりで入れたもの）を見たときに、そこで文字列が終わりだと思ってしまうことです。2 個目のクォートよりも後ろのものをどのように処理したらよいか、Python にはわかりません。

この問題は、文字列をダブルクォートで囲めば簡単に解決できます。

strings/single_in_double.cmd

```
>>> "that's better"
"that's better"
```

文字列にダブルクォートを挿入しなければならないときには、文字列をシングルクォートで囲みます。しかし、1 個の文字列に両方のクォートを挿入しなければならないときにはどうすればよいでしょうか。次のような方法は確かに可能です。

strings/adding_quotes.cmd

```
>>> 'She said, "That' + "'" + 's hard to read."'
```

しかし、もっとよい方法があります。Python に先ほどの式を入力すると、結果は次のようになります。

strings/adding_quotes_output.cmd

```
'She said, "That\'s hard to read."'
```

バックスラッシュとシングルクォートの組み合わせは、**エスケープシーケンス**（escape sequence）と呼ばれるものの 1 つです。エスケープシーケンスという名称は、Python の通常の構文規則をいったん「エスケープする」（逃れる）という意味から来ています。Python は、文字列の中でバックスラッシュを見かけると、次の文字が特別な意味を表すものと見なします。この場合なら、文字列の末尾ではなく、シングルクォートの文字そのものということです。バックスラッシュは、エスケープシーケンスの先頭を知らせるので、**エスケープ文字**（escape character）と呼ばれています。

表 3.1 に示すように、Python は、複数のエスケープシーケンスを認識します。これらがどのように使われるかを理解するためには、マルチライン（複数行）文字列と表示の 2 つの観念を知る必要があります。

表3.1 エスケープシーケンス

エスケープシーケンス	意味
\n	行の末尾（改行）
\\	バックスラッシュ
\'	シングルクォート
\"	ダブルクォート
\t	タブ

3.3 マルチライン文字列

シングルクォートやダブルクォートを使って文字列を作った場合、文字列全体は 1 行に収まっていなければなりません。

文字列を複数行に分割しようとすると、次のようになります。

strings/multi1.cmd

```
>>> 'one
Traceback (most recent call last):
  File "<string>" , line 1, in <string>
Could not execute because an error occurred:
  EOL while scanning single-quoted string: <string>, line 1, pos 4:
  'one
```

EOL は「end of line」(行末) という意味ですから、このエラーメッセージは、文字列の末尾を検出していないのに行末を検出してしまったという意味になります。

複数行の文字列を使うには、文字列の前後に 1 個ではなく 3 個のシングル / ダブルクォートを付けます。こうすると、文字列は何行までででも延ばしていけます。

strings/multi2.cmd

```
>>> '''one
... two
... three'''
'one\ntwo\nthree'
```

私たちの入力が新しい行を開始するたびに、Python が作る文字列に \n というシーケンスが入っていることに注意して下さい。実際には、3 大 OS は行末を示すために異なる文字を使っていますが、この文字を**改行** (newline) と呼びます。Linux と Mac OS X では改行は '\n' の 1 字ですが、バージョン 9 までの Mac OS は '\r' を使っていました。そして、Windows では、'\r\n' の 2 字で改行を表します。

Python は、改行を示すために常に \n の 1 文字を使います。Windows のように、実際には異なるものを改行として使っているシステムでも、\n を使います。これを文字列の**正規化** (normalizing) と呼びます。Python は、どのようなマシンでプログラムを実行する場合でも、同じプログラムを書けるようにするために、正規化を行っています。

3.4 print

今までは、同時に表示できるのは 1 個の変数または式の値だけでした。しかし、実際のプログラムでは、複数の変数の値など、より多くの情報を表示すべき場合が多々あります。そのようなときには、print 文を使います。

strings/print3.cmd

```
>>> print 1 + 1
2
>>> print "The Latin 'Oryctolagus cuniculus' means 'domestic rabbit'."
The Latin 'Oryctolagus cuniculus' means 'domestic rabbit'.
```

最初の文は、今までの数値の例から予想できる処理をしていますが、2 番目のものは今までの文字列の例と少し異なります。文字列の前後のクォートを取り除き、文字列の表現ではなく、内容を表示しているのです。次の例は、2 つの違いをはっきりと示しています。

strings/print4.cmd

```
>>> print 'In 1859, Charles Darwin revolutionized biology'
In 1859, Charles Darwin revolutionized biology
```



```
(1859年、チャールズ・ダーウィンは、生物学と)
>>> print 'and our understanding of ourselves'
and our understanding of ourselves
(私たちの自分自身についての理解に革命的な変革をもたらした。)
>>> print 'by publishing "On the Origin of Species".'
by publishing "On the Origin of Species".
(『種の起源』を出版することによって。)
```

そして次の例は、Python が文字列を表示するときには、バックスラッシュ付きの表現ではなく表現が表している値として文字列内のエスケープシーケンスを表示することを示しています。

strings/print5.cmd

```
>>> print 'one\ttwo\nthree\tfour'
one     two
three   four
```

このサンプルは、タブ文字の \t を使えば、値を揃えて表示できることを示しています。print 文は、表示対象の項目をカンマで区切ったリストを引数として取り、専用の行にそれらを並べて表示します。引数値なしの print は、単純に空行を表示します。リスト内では、さまざまな型を混ぜて使うことができます。個々の値の間には、Python がかならず 1 個のスペースを挿入します。

strings/print_var.cmd

```
>>> area = 3.14159 * 5 * 5
>>> print "その円の面積は", area, "平方センチです。"
その円の面積は 78.53975 平方センチです。
```

3.5 整形されたprint

Python のデフォルト出力ルールでは満足できないことがあります。そのようなときには、**整形文字列** (format string) を使って出力形式を正確に指定することができます。

strings/print.cmd

```
>>> print "その円の面積は%f平方センチです。" % area
その円の面積は78.539750平方センチです。
```

上の文で、%f は**変換指定子** (conversion specifier) で、area 変数の値をどこに挿入すべきかを示しています。他の変換指定子としては、文字列値の挿入を指示する %s や、整数値の挿入を指示する %d があります。% の後ろの文字は、**変換型** (conversion type) と呼ばれます。

文字列と変数の間の % も、多重定義演算子の例です。以前は剰余 (モジュロ) 演算子として % を使いましたが、ここでは文字列整形演算子として使っています。この演算子は、3 + 5 の + が 3 の値に変更を加えないのと同じように、左辺の文字列には変更を加えません。代わりに、文字列整形演

算子は、新しい文字列を返します。

文字列整形演算子を使えば、複数の値を同時にレイアウトできます。たとえば、次のコードは、float と int を同時にレイアウトしています。

strings/print2.cmd

```
>>> rabbits = 17
>>> cage = 10
>>> print "%d番の小屋には%f羽のうさぎがいます。" % (cage, rabbits)
10番の小屋には17.000000羽のうさぎがいます。
```

以前に言ったように、print は、文字列の末尾に自動的に改行を挿入しますが、そのような動作が望ましくない場合もあります。たとえば、複数のデータを1行の中で区切って表示したい場合などです。このようなときには、print 文の末尾にカンマを付けると、改行は挿入されません。

strings/print_multiline2.cmd

```
>>> print rabbits,
17>>>
```

3.6 ユーザー入力

前章では、一部の組み込み関数を試してみました。組み込み関数としては他に、キーボードから1行分のテキストを読み出すraw_inputなどもあります。「raw」(生)という表現が使われているのは、たとえ数値のように見えても、ユーザーが入力したものを文字列として返すからです。

strings/user_input.cmd

```
>>> line = raw_input()
ガラパゴス諸島
>>>  print  line
ガラパゴス諸島
>>> line = raw_input()
123
>>> print line * 2
123123
```

ユーザーが数値を入力することになっている場合には、int か float を使って文字列から必要な型に変換しなければなりません。

strings/user_input2.cmd

```
>>> value = raw_input()
123
>>> value = int(value)
>>> print value * 2
```



```
246
>>> value = float(raw_input())
ガラパゴス諸島
Traceback (most recent call last):
  File "<stdin>" , line 1, in <module>
ValueError: invalid literal for float(): ガラパゴス諸島
```

raw_input には、文字列引数を与えることもできます。この文字列は、ユーザーに入力を促すプロンプトとして使われます。

strings/raw_input_param.cmd

```
>>> name = raw_input("名前を入力して下さい: ")
名前を入力して下さい: ダーウィン
>>> print name
ダーウィン
```

3.7 まとめ

この章では、次のことを学びました。

- Python は、文字列型の str を使って文字の並び(シーケンス)としてのテキストを表現します。
- 文字列は、テキストの前後にシングルクォートかダブルクォートを追加して作ります。同じ種類のクォートを3つ並べたトリプルクォートを使えば、複数行のテキストを作ることもできます。
- 改行やタブなどの特殊文字は、バックスラッシュを先頭に置くエスケープシーケンスを使って表現することができます。
- print 文を使えば、画面に値を表示できます。また、raw_input を使えば、ユーザーが入力した文字列を取り込むことができます。

3.8 練習問題

練習問題で自分の力を試してみましょう。

1. 次の式は、それぞれどのような値を返すでしょうか。Python シェルに式を入力して、答えを確かめて下さい。
   a) 'Comp' 'Sci'
   b) 'Computer' + ' Science'
   c) 'H2O' * 3
   d) 'CO2' * 0
2. 適切なクォート(シングル、ダブル、トリプル)と必要な場合はエスケープシーケンスを使って、

次のテキストを Python 文字列として表現して下さい。

a) They'll hibernate during the winter.
b) "Absolutely not," he said.
c) "He said, 'Absolutely not,'" recalled Mel.
d) hydrogen sulfide
e) left\right

3. トリプルクォートではなく、シングルクォートかダブルクォートを使って次の文字列を書き直して下さい。

```
'''A
B
C'''
```

4. 組み込み関数の len を使って、空文字列の長さを調べて下さい。

5. それぞれ 3、12.5 を参照する x、y 変数があるとき、print を使って次のメッセージを表示して下さい。ただし、メッセージに含まれる数値については、x、y 変数を使って表示しなければならないものとします。

a) The rabbit is 3.
b) The rabbit is 3 years old.
c) 12.5 is average.
d) 12.5 * 3
e) 12.5 * 3 is 37.5.

6. 「3.5 整形された print」では、% 演算子を使って文字列を整形する方法を紹介しました。次の出力を得るためには、どのような書式文字列を使えばよいか答えて下さい。

a) "____" % 34.5 ⇒ "34.50"
b) "____" % 34.5 ⇒ "3.45e+01"
c) "____" % 8 ⇒ "0008"
d) "____" % 8 ⇒ "8 "

7. raw_input を使ってユーザーに数値の入力を促し、入力された数値を float 型の num 変数に格納して、num を表示して下さい。

8. Python で 2 つの文字列を隣り合わせで入力すると、Python は自動的に 2 つの文字列を連結します。

```
>>> 'abc' 'def'
'abcdef'
```

しかし、同じ文字列を変数に格納してから変数を横に並べると、構文エラーになります。

```
>>> left = 'abc'
>>> right = 'def'
>>> left right
  File "<stdin>" , line 1
left right
 ^
SyntaxError: invalid  syntax
```


Python がこれを認めないのはなぜだと思いますか。

9. 文字列に負数を掛け合わせたときに、空文字列を返すのではなく、エラーを起こすべきだと考えている人々がいます。彼らがそのように考える理由を説明しなさい。また、この意見に賛成、反対のどちらでもかまいませんから、自分の考えの理由を説明して下さい。