文字列とは？

- プログラミングというと数学という印象を持つかもしれないが、実際にはほとんどのプログラマは、**数値よりも文字列を操作することが多い。**
- 文字列はシーケンス（連なり、列、並び）の一種で、**文字のシーケンス。**
- 文字には、アルファベット、数字、記号、符号のほか、スペースや改行などの指示といったものまで含まれる。
- 文字列はシーケンスなので、値を部分的に処理することができる。
- Pythonでは文字列は**イミュータブル**なオブジェクト。
- イミュータブルとは、**不変**という意。
- **イミュータブルなオブジェクト**とは、**作成後にその状態を変えることができないオブジェクト**のことをいう。
- 文字列の一部を別の文字列としてコピーして、**元の文字列を変更したかのような結果を得ること**はできる。

文字列の演算（データ型の表記：str）

```
>>> 'Caesar'
'Caesar'
>>> "Brutus"
'Brutus'
>>> "'Nay!' said the naysayer. 'Neigh?' said the horse."
"'Nay!' said the naysayer. 'Neigh?' said the horse."
>>> 'A "two by four" is actually 1 1/2" x 3 1/2".'
'A "two by four" is actually 1 1/2" x 3 1/2".'
>>> '''Caesar'''
'Caesar'
>>> """You too, Brutus?"""
'You too, Brutus?'
>>> poem = """There was a Young Lady of Norway,
... Who casually set in a doorway;
... When the door squeezed her flat,
... She exclaimed, "What of that?"
... This courageous Young Lady of Norway."""
>>> poem
'There was a Young Lady of Norway,¥nWho casually set in a doorway; ¥nWhen the door squeezed her flat, ¥nShe exclaimed, "What of that?" ¥nThis courageous Young Lady of Norway.'
>>> print(poem)
There was a Young Lady of Norway,
Who casually set in a doorway;
When the door squeezed her flat,
She exclaimed, "What of that?"
This courageous Young Lady of Norway.
>>> poem = 'There was a Young Lady of Norway,
  File "<stdin>", line1
    poem = 'There was a Yong Lady of Norway,
                    ^
SyntaxError: unterminated string literal (detected at line 1)
```

文字列の演算（データ型の表記：str）

```
19 >>> poem2 = """I do not like thee,
   Doctor Fell.
   ...    The reason why, I cannot tell.
   ...    But this I know, and know full
   well:
   ...    I do not like thee, Doctor Fell.
   ... """
20 >>> print(poem2)
   I do not like thee, Doctor Fell.
       The reason why, I cannot tell.
       But this I know, and know full
   well:
       I do not like thee, Doctor Fell.

21 >>> poem2
   'I do not like thee, Doctor
   Fell.¥n    The reason why, I cannot
   tell.¥n    But this I know, and know
   full well:¥n    I do not like thee,
   Doctor Fell.¥n'
22 >>> print('You', "too,", """Brutus?""")
   You too, Brutus?
23 >>> print('You', "too,", """Brutus?""",
   sep='')
   Youtoo,Brutus?
24 >>> print('You', "too,", """Brutus?""",
   end='')
25 You too, Brutus?>>> ''
   ''
26 >>> ""
   ''
27 >>> ''''''
   ''
28 >>> """""""
   ''
29 >>>
   season='Spring,¥nSummer,¥nFall,¥nWinter.'
30 >>> print(season)
   Spring,
   Summer,
   Fall,
   Winter.
```

文字列の演算（データ型の表記：str）

```
>>> print('¥tabc')
        abc
>>> print('a¥tbc')
a       bc
>>> print('ab¥tc')
ab      c
>>> print('abc¥t')
abc
>>> "¥"Nay!¥" said the naysayer. ¥"Neigh?¥" said the horse."
'"Nay!" said the naysayer. "Neigh?" said the horse.'
>>> '¥'Nay!¥' said the naysayer. ¥'Neigh?¥' said the horse.'
"'Nay!' said the naysayer. 'Neigh?' said the horse."
>>> speech = 'This symbol: ¥¥ is backslash.'
>>> print(speech)
This symbol: ¥ is backslash.
```

目に見えないタブ文字がある。

```
>>> str(3.14)
'3.14'
>>> str(1.0e4)
'10000.0'
>>> str(True)
'True'
>>> 'Hello, ' + 'world!'
'Hello, world!'
>>> 'Hello, ' 'world!'
'Hello, world!'
>>> 'You too, ''Brutus?'
'You too, Brutus?'
>>> vowels = ( 'a'
..."e" """i"""
..."o" """u"""
...)
>>> vowels
'aeiou'
```

文字列の演算（データ型の表記：str）

```
>>> start = 'Go ' * 2 + 'Waseda¥n'
>>> middle = 'Hey ' * 3 + '¥n'
>>> end = 'Goodbye.'
>>> print(start + start + middle + end)
Go Go Waseda
Go Go Waseda
Hey Hey Hey
Goodbye.
>>> letters =
'abcdefghijklmnopqrstuvwxyz'
>>> letters[0]
'a'
>>> letters[1]
'b'
>>> letters[-1]
'z'
>>> letters[-2]
'y'
>>> letters[25]
'z'
>>> letters[5]
'f'
```

改行の前に「*」の演算によって文字列が追加されている。

```
>>> letters[100]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> name = 'Cython'
>>> name[0] = 'P'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not
support item assignment
```

文字列オブジェクトはイミュータブルなので直接的に値を書き換えることができない。この対処法は後ほど解説。

スライスによる部分文字列の取り出し

- 文字列はシーケンス。シーケンスからは部分を取り出せる（スライス）。
- スライスは、角括弧と先頭オフセット(start)、末尾オフセット(end)、ステップ(step）で定義する。
- 省略記法もある。
- スライスには、startから**end**の**1字手前までの文字が含まれる**。
  - [:]は、先頭から末尾までのシーケンス全体を抽出する。
  - [start:]は、startオフセットから末尾までのシーケンスを抽出。
  - [:end]は、先頭からend – 1オフセットまでのシーケンスを抽出。
  - [start:end]は、startオフセットからend – 1オフセットまでのシーケンスを抽出。
  - [start:end:step]は、step文字ごとにstartオフセットからend – 1オフセットまでのシーケンスを抽出。
- オフセットはゼロから右に向かって0、1・・・となり、末尾から左に向かって–1、–2・・・となる。*start*を指定しないと0からとなる。

文字列の演算（データ型の表記：str）

```
61 >>> letters =
   'abcdefghijklmnopqrstuvwxyz'
62 >>> letters[:]
   'abcdefghijklmnopqrstuvwxyz'
63 >>> letters[20:]
   'uvwxyz'
64 >>> letters[10:]
   'klmnopqrstuvwxyz'
65 >>> letters[12:15]
   'mno'
66 >>> letters[-3:]
   'xyz'
67 >>> letters[18:-3]
   'stuvw'
68 >>> letters[-6:-2]
   'uvwx'
69 >>> letters[::7]
   'ahov'
70 >>> letters[4:20:3]
   'ehknqt'
71 >>> letters[19::4]
   'tx'
```

文字列全体の指定。

オフセット20から末尾まで。
先頭から21字目とも解釈。

オフセット10から末尾まで。

オフセット12から14まで。
末尾オフセットは含まれない。

最後の3文字をスライス。

オフセットは -4 のwまで
になることに注意。

末尾から6字目を先頭として
末尾から3字目までスライス。

step7で全体からスライス。

```
72 >>> letters[:21:5]
   'afkpu'
73 >>> letters[-1::-1]
   'zyxwvutsrqponmlkjihgfedcba'
74 >>> letters[::-1]
   'zyxwvutsrqponmlkjihgfedcba'
75 >>> letters[-50:]
   'abcdefghijklmnopqrstuvwxyz'
76 >>> letters[-51:-50]
   ''
77 >>> letters[:70]
   'abcdefghijklmnopqrstuvwxyz'
78 >>> letters[70:71]
   ''
79 >>> name
   'Cython'
80 >>> 'P' + name[1:]
   'Python'
81 >>> len(letters)
   26
82 >>> empty = ''; len(empty)
   0
```

ステップに負数を指定す
ると、逆方向にステップ。

文字列を直接的に書き換えること
ができないことへの対処法。

１行に連続して命令を書く方法。

Pythonの構文規則の補足

```
>>> a = 3; print(a)
3
```

- 通常、2行に分けて書く命令を1行で書いた例。
- 事情がない限り、複数命令を1行で書くのは非推奨。
- Pythonでは命令の終わりにセミコロン；は不要。
- 他のプログラミング言語では；が必須な場合あり。
- 先のスライド末尾のように事情がある場合はこの限りではないが、一般的なPythonプログラムでこの書き方は見かけない。

```
>>> a = 1
>>>   print(a)
  File "<stdin>", line 1
    print(a)
IndentationError: unexpected indent
```

- 字下げ（インデント）を意図なくつけるとエラー。
- 命令の範囲をブロックという。
- Pythonではブロックをインデントで表現する。
- 上記のように書くと a = 1 の命令が print(a) に続くかのような判定となり、矛盾してエラー発生。
- 他言語ではブロックを「波括弧{}」やbeginとendの予約語で表現することがある。

```
>>> a =
  File "<stdin>", line 1
    a =
      ^
SyntaxError: invalid syntax
>>> a = ¥
... 3
>>> b = (
... 1)
>>> a
3
>>> print(b)
1
>>> print(
... a
... )
3
>>> print(
... a    )
3
>>>
```

改行が必要な場合は、バックスラッシュまたは¥、括弧を併用。

特殊な代入、f文字列、int型変換の補足

```
>>> two = deux = zwei = 2
>>> two
2
>>> deux
2
>>> zwei
2
>>> a, b = 1, 2
>>> a
1
>>> b
2
>>> a, b = b, a
>>> a
2
>>> b
1
>>> place = 'Waseda park'
>>> time = 8
>>> f'{place} at {time}am.'
'Waseda park at 8am.'
```

同時に複数の変数に代入

並列に複数の変数に代入

値の入れ替え（参照先の付け替え）

```
>>> int('31'); int('-31'); int('+31')
31
-31
31
>>> int('1_000_000')
1000000
>>> int('10',2);int('10',16)
2
16
>>> int('10', 22)
22
>>> int(12345)
12345
>>> int('31 bottles of beer')
  (略)ValueError (略)
>>> int('')
  (略)ValueError (略)
>>> int('31.4')
  (略)ValueError (略)
>>> int('3.0e4')
  (略)ValueError (略)
```

文字列が10進数以外の整数を表す場合、基数を指定できる。int関数は基数に応じた変換を行い10進数の数値を返す。

まとめ

- **演算**とは単に四則計算ではなく**四則演算**、**論理演算**、**比較演算**、**代入演算**などを意味する語。
- 演算を行うには、**値**と**演算子**が必要である。
- 値はデータ型に応じて演算規則が分かれており、基本的なものとして**ブール値**や**整数**、**浮動小数点数**、**文字列**がある。
- 演算子には、+や–などの**算術演算子**、andやorなどの**論理演算子**、＝＝などの比較演算子、＝や+＝などの**代入演算子**がある。
- 演算では**型、リテラル、変数、関数などの構文規則を厳守**する必要あり。
- 文字列は**イミュータブルなオブジェクト**である。
- **部分文字列を利用**することにより**文字列の編集が可能**となる。
- **文字列はPythonシーケンスの一種**であり、**スライス構文**が利用可能。
- データ型は変換可能であるが例外に注意。

次回予告　**分岐処理**