Python学習キット初級編

REPL・スクリプト

REPL

PythonをREPL(Read Eval Print Loop)として使用するにはコマンド`python`を実行する。

```
$ python
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] o
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

簡単な電卓として使用してみる。

```
>>> x = 10
>>> y = 3
>>> x + y
13
>>> x * y
30
>>> z = 2.0
>>> y - z
1.0
```

スクリプト

Pythonのスクリプトは`ファイル名.py`という名前でファイルとして保存することが出来る。

```
$ echo 'print("Hello, world")' > hello.py
```

保存したスクリプトは`python ファイル名.py`で実行することが出来る。

```
$ python hello.py
Hello, world
```

See Also

- https://docs.python.org/ja/3.10/tutorial/interpreter.html

文法・構文

変数

変数には`=`で値を代入する。

```
x = 1
name = "Takeuchi"
```

数値リテラル

```
x = 1       # 整数
y = 2.0     # 浮動小数点数
z = 3 + 5j  # 複素数
```

演算

```
>>> 2 + 2      # 和
4
>>> 5 - 3      # 差
2
>>> 1.5 * 3    # 積
4.5
>>> 6 / 2      # 商
3.0
>>> 9 // 4     # 整数除算
2
>>> 9 % 4      # 余り
1
>>> 3 ** 2     # べき乗
9
>>> (5 - 2) * 3
9
>>> 4 | 2      # ビット和
6
>>> 7 & 2      # ビット積
2
>>> 1 << 2     # 左へビットシフト
4
>>> 4 >> 1     # 右へビットシフト
2
```

異なる型同士の演算はPythonが上手く処理してくれる。

See Also

- https://docs.python.org/ja/3.10/library/stdtypes.html#numeric-types-int-float-complex

文字列リテラル

文字列は単引用符(`'...'`)または二重引用符(`"..."`)で囲う。

```
hello = "Hello"
bye = 'Good bye'
```

三連引用符(`"""..."""` , `'''...'''`)を使って複数行にもできる。

```
intro = """\
Hello, I'm Monty Python.
Nice to meet you.
"""
```

簡単な文字列操作

```
>>> "Hello" + ",world"  # 結合
'Hello,World'
>>> "Hello" * 3         # 反復
'HelloHelloHello'
word = "Python"
>>> word[0]             # インデックス
'P'
>>> word[-1]            # 最後の要素を取得
'n'
>>> word[1:3]           # 部分列を取得（スライス）
'yt'
>>> word[:4]            # スライスの先頭を省略
'Pyth'
>>> word[2:]            # スライスの末尾を省略
'thon'
>>> word[::2]           # 一つ飛ばしでスライス
'Pto'
>>> len(word)           # 長さ
6
```

文字列の接頭辞

```
>>> "hello\n"
'hello\n'
```

```
>>> r"hello\n"       # エスケープしない
'hello\\n'
>>> name = "Bob"
>>> f"hello, {name}" # フォーマット
'hello, Bob'
>>> text = b"hello"  # UTF-8でエンコード
>>> type(text)       # 型はバイト型
<class 'bytes'>
```

See Also

- https://docs.python.org/ja/3.10/library/stdtypes.html#text-sequence-type-str

データ構造

タプル

要素をコンマ(`,`)で区切るとタプル型オブジェクトを作ることができる。
タプルは複数の値をまとめて扱いたい時に使用する。

```
pair = 1, 3
person = ("Alice", 20, 2)  # ()で囲っても同じ
```

簡単なタプルの操作

```
>>> person = ("Alice", 20, 2)
>>> person[0]                  # インデックス
'Alice'
>>> person[1:]                 # スライス
(20, 2)
>>> name, age, grade = person  # パース
>>> name
'Alice'
>>> age
20
>>> grade
2
>>> person[0] = "Bob"          # タプルの要素を書き換えられない。
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

See Also

- https://docs.python.org/ja/3.10/library/stdtypes.html#sequence-types-list-tuple-range

リスト

要素をコンマ(`,`)で区切って角括弧(`[]`)で囲うとリスト型オブジェクトを作ることができる。

```
primes = [2, 3, 5, 7, 11]
names = [                                   # 複数行に書く場合
    "Alice",
    "Bob",
    "Charlie"
]
squares = [i**2 for i in range(1, 10)]  # 内包表記
values = list()                             # 空のリストを作成
```

簡単なリスト操作

```
>>> primes = [2, 3, 5, 7, 11]
>>> primes[2]               # インデックス
5
>>> primes[1:4]             # スライス
[3, 5, 7]
>>> primes + [13, 17, 19]  # 連結
[2, 3, 5, 7, 11, 13, 17, 19]
>>> primes.append(13)       # 追加
>>> primes
[2, 3, 5, 7, 11, 13]
>>> primes[0] = 0           # 要素の変更
>>> primes
[0, 3, 5, 7, 11, 13]
```

See Also

- https://docs.python.org/ja/3.10/tutorial/datastructures.html#more-on-lists
- https://docs.python.org/ja/3.10/library/stdtypes.html#sequence-types-list-tuple-range

集合

要素をカンマ(`,`)で区切って波括弧(`{}`)で囲うと集合型オブジェクトを作ることができる。
集合型オブジェクトの要素は重複せず、順序も存在しない。

```
a = {2, 4, 6, 8, 10}
b = {i * 2 for i in a}  # 内包表記
c = set()               # 空の集合を作成
```

簡単な集合操作

```
>>> a = {2, 4, 6, 8, 10}
>>> b = {3, 6, 9, 12, 15}
>>> a.add(12)          # 要素を追加
>>> a
{2, 4, 6, 8, 10, 12}
>>> b.remove(15)       # 要素を削除
>>> b
{3, 6, 9, 12}
>>> 8 in a             # 包含判定
True
>>> 8 not in b
True
>>> a | b              # 和集合
{2, 3, 4, 6, 8, 9, 10, 12}
>>> a & b              # 積集合
{12, 6}
>>> a - b              # 差集合
{8, 2, 10, 4}
>>> a ^ b              # 対称差集合
{2, 3, 4, 8, 9, 10}
```

See Also

- https://docs.python.org/ja/3.10/library/stdtypes.html#set-types-set-frozenset

辞書

キー:値のペアの集合を波括弧(`{}`)で囲うことで辞書型オブジェクトを作成できる。

```
age = {
    "Alice": 20,
    "Bob": 25,
    "Charlie": 30
}
double = {i: 2*i for i in range(5)}  # 内包表記
key_to_value = dict()                # 空の辞書を作成
```

簡単な辞書操作

```
>>> age = {"Alice": 20, "Bob": 25, "Charlie": 30}
>>> age["Alice"]        # 値の取得
20
>>> age["Bob"] = 26     # 値の更新
>>> age
{'Alice': 20, 'Bob': 26, 'Charlie': 30}
>>> age["Daniel"] = 32  # 新しいペアを追加
>>> age
```

```
{'Alice': 20, 'Bob': 26, 'Charlie': 30, 'Daniel': 32}
>>> del age["Charlie"]  # ペアを削除
>>> age
{'Alice': 20, 'Bob': 26, 'Daniel': 32}
>>> "Eric" in age       # キーの包含判定
False
>>> age["Eric"]         # キーが存在しなければエラーになる
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Eric'
```

See Also

- https://docs.python.org/ja/3.10/library/stdtypes.html#mapping-types-dict

制御フロー

真理値判定

if文

```
x = input("Enter an ingeger: ")
if x < 5:
    print("It is less than 5")
elif x == 5:
    print("It is 5")
else:
    print("It is greater than 5")
```

for文

```
nums = [1, 2, 3, 4, 5]
for n in nums:  # シーケンスの各要素についてイテレート
    print(n ** 2)
```

```
string = "Hello, world"
replaced = ""
for c in string:  # 文字列の各文字についてイテレート
    if c == "o":
        replaced += ""
    else:
        replaced += c
print(replaced)
```

```
nums = [1, 2, 3, 4, 5]
x = input("Enter an integer: ")
for n in nums:
    if n == x:
        print(f"{x} is in the list")
        break
else:  # for文が全て回った場合
    print(f"{x} is not in the list")
```

```
age = {"Alice": 20, "Bob": 25, "Charlie": 30}
for key in age:  # 辞書のキーをイテレート
    print("name:", key)
    print("age:", age[key], "\n")
```

See Also

- https://docs.python.org/ja/3.10/tutorial/controlflow.html#break-and-continue-statements-and-else-clauses-on-loops

while文

```
i = 0
while True:
    print(i)
    i += 1
    if i < 5:
        break
```

関数

構文と呼び出し

関数は`def`キーワードから始まり、関数名と仮引数を丸括弧(`()`)で囲んだリストを続ける。
実行文はインデントされたブロックに書く。
戻り値は`return`キーワードに続けて書く。

```
>>> def function(x, y, z):
...     return x * y + z
...
>>> function(3, 3, 5)       # 引数を指定して呼び出し
14
>>> function(x=3, y=3, z=5) # 引数名=値でもOK (キーワード引数)
14
```

```
>>> function(z=5, x=3, y=3) # 順番を入れ替えてもOK
14
>>> function(3, 3, z=5)     # キーワード引数は位置引数より後ろ
14
>>> function(z=5, 3, 3)     # これはダメ
File "<stdin>", line 1
    function(z=5, 3, 3)
                     ^
SyntaxError: positional argument follows keyword argument
```

デフォルト引数

仮引数のデフォルト値は仮引数=値で指定するここができる。

```
>>> def function(x, y=3, z=5):
...     return x * y + z
...
>>> function(3)
14
>>> function(3, 4)
17
>>> function(3, z=4)
13
```

可変長引数・キーワード引数

`*引数名`で可変長引数をタプル型として受け取ることができる。また、`**引数名`でキーワード引数を辞書型として受け取ることができる。

```
>>> def function(pos, *args, **kwargs):
...     print("位置引数:", pos)
...     print("可変長引数:", args)
...     print("キーワード引数:", kwargs)
...
>>> function("a")
位置引数: a
可変長引数: ()
キーワード引数: {}
>>> function("a", "b", "c")
位置引数: a
可変長引数: ('b', 'c')
キーワード引数: {}
>>> function("a", "b", "c", text="hello")
位置引数: a
可変長引数: ('b', 'c')
キーワード引数: {'text': 'hello'}
```

逆に、`*`を使えばリストやタプルから引数をアンパックすることができる。
同様に、`**`で辞書もキーワード引数としてアンパックできる。

```
>>> args = [1, 2, 3]
>>> kwargs = {"name": "Alice", "age": 20}
>>> function("a", *args, **kwargs)
位置引数: a
可変長引数: (1, 2, 3)
キーワード引数: {'name': 'Alice', 'age': 20}
```

See Also

- https://docs.python.org/ja/3.10/tutorial/controlflow.html#more-on-defining-functions

組み込み関数

よく使用する組み込み関数を挙げる。

print()

テキストストリームにオブジェクトをプリントする。

```
>>> print("hello")
hello
>>> print("hello", "world")              # 複数のオブジェクトを同時にプリント
hello world
>>> print("hello", "world", sep=", ")    # セパレータを変更
hello, world
>>> print("hello", "world", end="!\n")   # 文末を変更
hello world!
```

int(), float(), bool(), str(), bytes()

各種型のオブジェクトを作成する。

```
>>> int(2.3)
2
>>> float(4)
4.0
>>> bool(0)
False
>>> bool(1)
True
>>> str(5)
'5'
>>> str(10.0)
'10.0'
>>> bytes("hello", encoding="utf-8")
b'hello'
```

tuple(), list(), set(), dict()

各種型のオブジェクトを作成する。

```
>>> base = [1, 1, 3, 5, 7]
>>> tuple(base)
(1, 1, 3, 5, 7)
```

```
>>> list(base)
[1, 1, 3, 5, 7]
>>> set(base)
{1, 3, 5, 7}
>>> dict(a=1, b=2)
{'a': 1, 'b': 2}
```

range()

Rangeオブジェクトを作成する。

```
>>> list(range(3))
[0, 1, 2]
>>> list(range(1, 5))
[1, 2, 3, 4]
>>> list(range(0, 10, 2))
[1, 3, 5, 7, 9]
>>> [i for i in range(1, 10) if i % 2 == 1]
[1, 3, 5, 7, 9]
```

len()

オブジェクトの長さを取得する。

```
>>> obj = list(range(5))
>>> obj
[0, 1, 2, 3, 4]
>>> len(obj)
5
>>> len("hello, world")
12
```

reversed(), sorted(), enumerate(), zip()

イテラブルオブジェクトを操作する。
戻り値はジェネレータ。

```
>>> iterable = [3, 1, 5, 6, 2]
>>> list(reversed(iterable))    # 順番を反転して生成する
[2, 6, 5, 1, 3]
>>> list(sorted(iterable))      # ソートして生成する
[1, 2, 3, 5, 6]
>>> list(enumerate(iterable))   # 添え字とのタプルを生成する
[(0, 3), (1, 1), (2, 5), (3, 6), (4, 2)]
```

```
>>> index = list(range(len(iterable)))
>>> list(zip(index, iterable))  # 複数のイテラブルの要素をタプルとして生成
[(0, 3), (1, 1), (2, 5), (3, 6), (4, 2)]
```

sum(), min(), max(), any(), all()

```
>>> iterable = list(range(5))
>>> sum(iterable)  # 合計を取得
10
>>> min(iterable)  # 最小値を取得
0
>>> max(iterable)  # 最大値を取得
4
>>> any(iterable)  # 論理和
True
>>> all(iterable)  # 論理積
False
```

open()

ファイルを開く。

```
>>> filename = "sample.txt"
>>> with open(filename, mode="w") as f:  # 書き込みモードで開く
...     f.write("hello, world")
...
12
>>> with open(filename, mode="r") as f:  # 読み取りモードで開く
...     text = f.read()
...
>>> print(text)
hello, world
>>> with open(filename, mode="a") as f:  # 追記モードで開く
...     f.write("Thank you")
...
9
>>> with open(filename) as f:            # デフォルトで読み取りモード
...     text = f.read()
...
>>> print(text)
hello, worldThank you
```