Python基礎

資料作成者：小寺　俊希

本講義のアウトライン

プログラミング入門

- プログラミングとは
- なぜPythonを使うのか

Python文法(基礎)

- 演算(算術演算 / 比較演算 / 論理演算)・変数
- 条件分岐・ループ・関数

Python文法(発展)

- クラス
- モジュール・ライブラリ

プログラミング入門

プログラミングとは
なぜPythonを使うのか

プログラミング入門

Python文法(基礎)

Python文法(発展)

プログラミング入門

Question：プログラミングとは？

Answer：

コンピュータに対する命令列(＝プログラム)を設計・構築するプロセス

すなわち

ある計算を行う手順を記述する作業

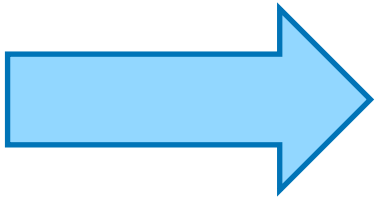
である

プログラミング入門

プログラミング

人間が理解できる言葉

1から10までの整数和を求めたい。
合計値を記憶することにして、
そこに1から10まで順に足せば
解が得られる。



機械が理解できる言葉

```
 Input: s = 0
        for n in range(10):
            s = s + n + 1
        s
Output: 55
```

プログラミング入門

Question：なぜPythonを使うのか？

Answer：

プログラムを簡単に書くことができる

- コンピュータに関する深い理解を必要としない
- 「退屈なことはPythonにやらせよう」

便利なツールが豊富に揃っている

- データ分析・機械学習のためのツールが多数開発されている

Python文法基礎

演算
変数
データ型
条件分岐
ループ
関数

プログラミング入門

Python文法(基礎)

Python文法(発展)

Python文法(基礎)

演算

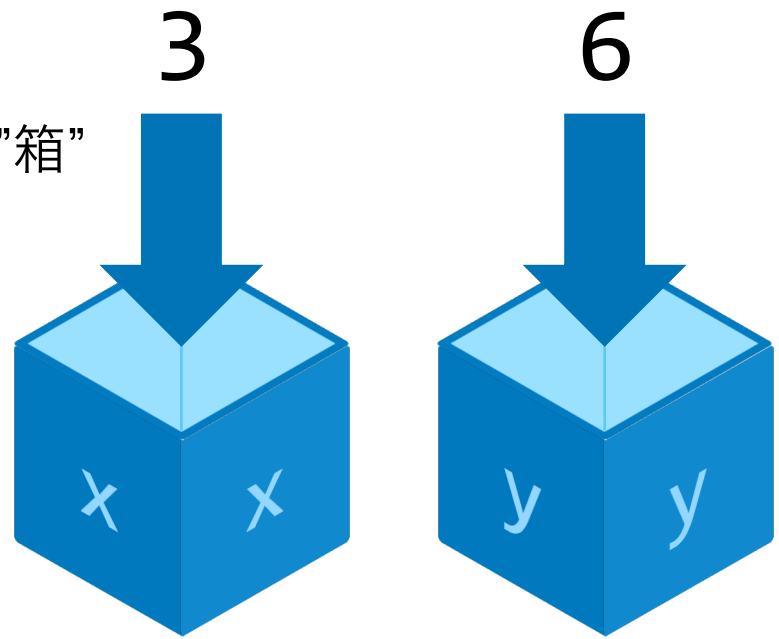
- 算術演算
  +, -, *(積), /(商), //(商), %(剰余)
- 比較演算
  >, <, >=, <=, ==, !=
- 論理演算
  and, or, not

Python文法基礎

変数

計算結果を一時的に保存しておくための”箱”

```
In: x = 1 + 2
    y = x + 3
    y

Out: 6
```



Python文法基礎

プログラム

- プログラムの実行は上から下へ順に進んでいく
- メモリーが好きなだけ使える
  「ちょっと便利な電卓」

```
# 1から5までの整数和

s = 0

a = 1 + 2
b = 3 + 4
c = 5
s = a + b
s = c + s

print(s)
```

Python文法基礎

データ型

- 整数 (int型)
- 小数 (float型)
- 真偽値 (bool型) `True / False`
- 文字列 (str型)
- リスト (list型) `[1, 2, 3, 4, 5]`
- タプル (tuple型) `(1, 2.0, “3”)`
- 辞書 (dict型) `{ “a”:1, “b”:2, “c”:3 }`

データ型(1) 整数 / int型

- 四則演算

```
 In: 2 * 3
Out: 6

 In: 7 // 3
Out: 2

 In: 7 % 3
Out: 1
```

- 比較演算

```
 In: 2 < 3
Out: True

 In: 2 == 3
Out: False

 In: 2 != 3
Out: True
```

Python文法基礎

データ型(2) 小数 / float型

- 四則演算

```
 In: 2.0 * 3
Out: 6.0

 In: 7.0 / 3.0
Out: 2.333333
```

- 比較演算

```
 In: 2.0 < 3
Out: True

 In: 0.5 + 0.5 == 1.0
Out: True

 In: 0.1 + 0.2 == 0.3
Out: False
```

Python文法基礎

データ型(3) 真偽値 / bool型

- 論理演算

```
 In: True and False
Out: False

 In: True or False
Out: True

 In: not(True)
Out: False
```

```
 In: not(0)
Out: True

 In: not(1)
Out: False

 In: 1 and 2
Out: 2
```

データ型(4) 文字列 / str型

- 文字列の結合

```
 In: “abc” + “def”
Out: “abcdef”
```

- 部分文字列の検索

```
 In: “abc” in “abcdef”
Out: True

 In: “xyz” in “abcdef”
Out: False
```

Python文法基礎

データ型(5) リスト / list型

- 要素へのアクセス

```
 In: x = [1, 2, 3]
     x[0]
Out: 1

 In: x = [1, 2, 3]
     x[-1] = 4
     x
Out: [1, 2, 4]
```

- 要素の検索

```
 In: 1 in [1, 2, 3]
Out: True

 In: “1” in [1, 2, 3]
Out: False
```

データ型(6) タプル / tuple型

- 要素へのアクセス

```
 In: x = (1, 2.0, “3”)
     x[0]
Out: 1
```

- 要素の検索

```
 In: 1 in (1, 2.0, “3”)
Out: True

 In: 2 in (1, 2.0, “3”)
Out: True
```

データ型(7) 辞書 / dict型

- 要素へのアクセス

```
 In: x = { ”a”:1, “b”:2, “c”:3 }
     x[“a”]
Out: 1

 In: x = { ”a”:1, “b”:2, “c”:3 }
     x[“c”] = 4
     x
Out: { ”a”:1, “b”:2, “c”:4 }
```

- キーの取得

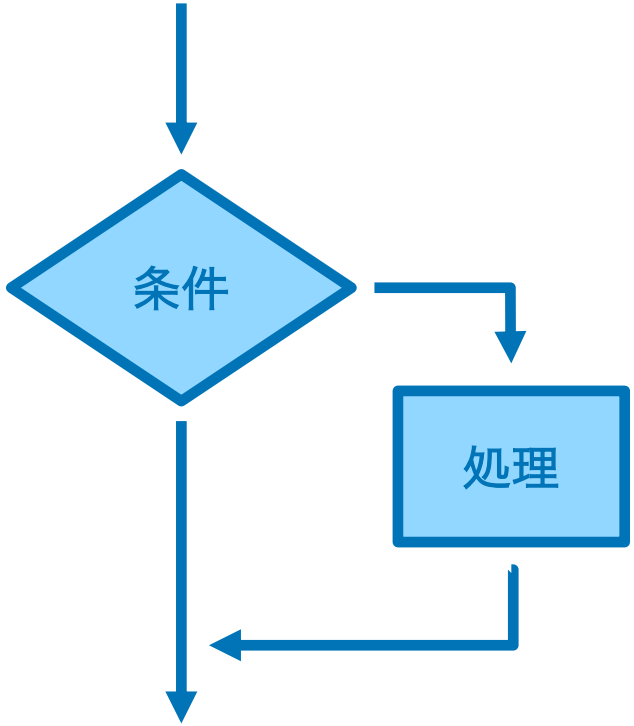
```
 In: x = { ”a”:1, “b”:2, “c”:3 }
     x.keys()
Out: dict_keys([“a”, “b”, “c”])
```

Python文法基礎

条件分岐

条件によって処理の流れを変えるための機能

```
if condition_a:
    # some procedure A
elif condition_b:
    # some procedure B
else:
    # some procedure C
```



Python文法基礎

条件分岐

```
In: x = -1
    if x < 0:
        y = - x
    elif x == 0:
        y = 0
    else:
        y = x
    y

Out: 1
```

```
In: x = -1
    if x <= 0:
        if x < 0:
            y = -x
        else:
            y = 0
    else:
        y = x
    y

Out: 1
```

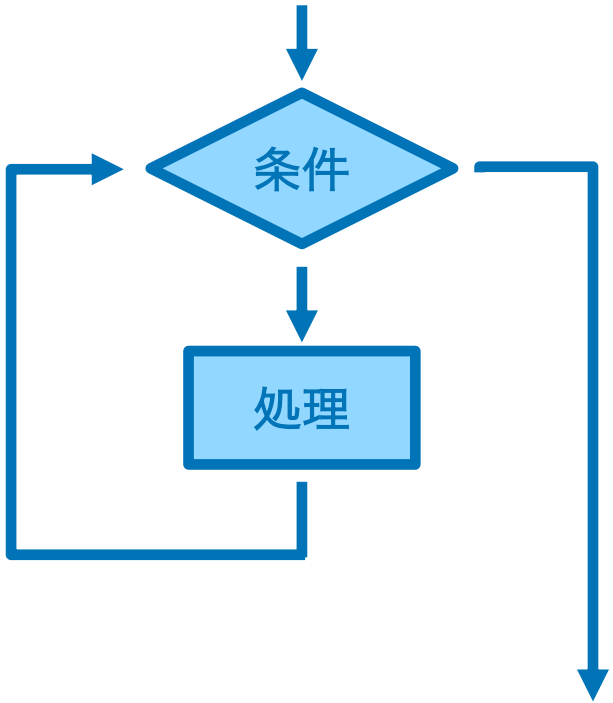
- インデントを行頭に入れることで
  ブロックとして認識される
- elif / else節は省略することができる
- ネストすることができる

Python文法基礎

ループ

同様の処理を複数回実行するための機能

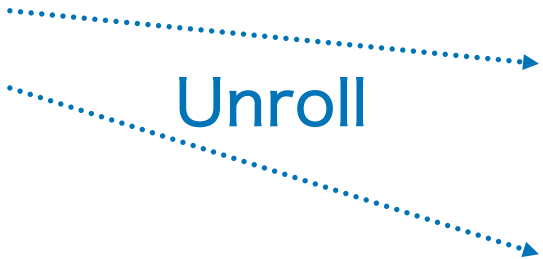
```
for loop_variable in iterable_object:
    # some procedure

while condition:
    # some procedure
```



Python文法基礎

ループ

```
In: s = 0
    for n in [1, 2, 3, 4, 5]:
        s = s + n
    s

Out: 15
```



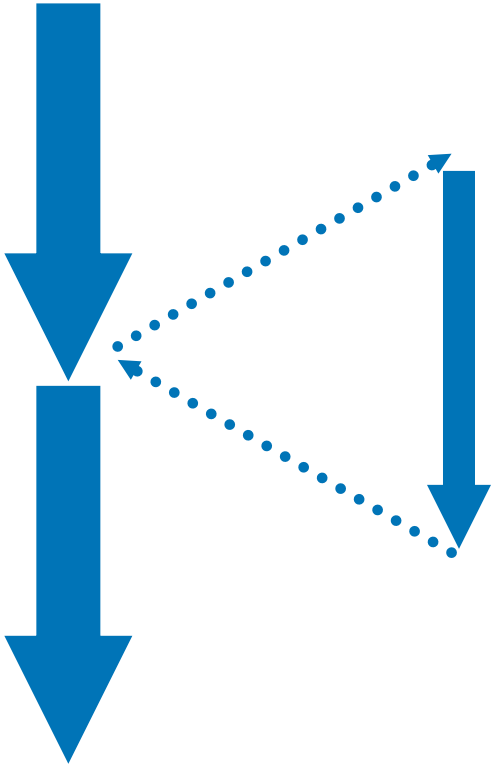

```
In: s = 0

    s = s + 1
    s = s + 2
    s = s + 3
    s = s + 4
    s = s + 5

    s

Out: 15
```

Python文法基礎

関数

処理に名前をつけ、
それを再利用できるようにするための機能

```
def function_name(arguments):
    # some procedure
```



Python文法基礎

関数

```
In: def f(x):
        y = 2 * x
        return y

    f2 = f(2)
    f3 = f(3)
    z = f2 + f3
    z

Out: 10
```

Expand

```
In: x2 = 2
    y2 = 2 * x2
    f2 = y2

    x3 = 3
    y3 = 2 * x3
    f3 = y3

    z = f2 + f3
    z

Out: 10
```

Python文法基礎

プログラム

- 条件分岐・ループ・関数を使うことで
  任意の場所にジャンプすることができる
- 数値データ以外もサポートしている
  「かなり便利な電卓」

```
# 1から5までの整数和

def f(x):
    return x + 1

s = 0

for i in [1, 2, 3, 4, 5]:
    d = f(i)
    s = s + d

if (s > 10):
    print(“s > 10”)
else:
    print(“s <= 10”)
```

Python文法
発展

オブジェクト指向
クラス
モジュール
ライブラリ

プログラミング入門

Python文法(基礎)

Python文法(発展)

オブジェクト指向プログラミング

オブジェクトを中心に設計するというプログラミングパラダイム

- プログラム中のデータは単なるデータではなく
  それに対するメソッドが定義されたオブジェクトである
- オブジェクトのデータに対する操作はメソッドを呼び出すことで間接的に行う
  (オブジェクトの外部から直接データを扱わない)
- Pythonはオブジェクト指向プログラミング言語である

Python文法応用

オブジェクト指向プログラミング

Object-Oriented

```
 In: lst = [1, 2, 3]
     lst.remove(1)
     lst
Out: [2, 3]
```

Data-Oriented

```
 In: def remove(lst, element):
         # remove element from lst

     lst = [1, 2, 3]
     remove(lst, 1)
     lst

Out: [2, 3]
```

Python文法応用

クラス

新しいデータ型を定義するための機能

- オブジェクトの設計図になる

```
In: class Counter:
        def __init__(self):
            self.cnt = 0
        def count(self):
            self.cnt = self.cnt + 1
        def get(self):
            return self.cnt
        def set(self, n):
            self.cnt = n

    x = Counter()
    x.count()
    x.count()
    x.get()

Out: 2
```

Python文法応用

モジュール

プログラムを再利用可能にするための機能

- 大きなプログラムを複数の小さなプログラムに分割して管理できる
- 他の人が開発したプログラムを簡単に利用できる

```
def f(x):
    return 2 * x

def g(x):
    return 3 * x
```

functions.py

```
import functions

f = functions.f(2)
g = functions.g(2)
h = f + g
h
```

program.py

Python文法応用

ライブラリ

再利用可能な形でまとめられたモジュール群

- Pythonはデータ分析・機械学習のために開発されているライブラリが多数ある
- 仕様を知るだけで便利な道具が使い放題になる！
- この講座では様々なライブラリの使い方を学習する
  (Numpy, Pandas, Matplotlib, Scikit-learn, ...)

Python文法応用

Python学習のキーワード

- 無名関数 (lambda式)
- map関数
- リスト内包表記
- 条件式 (三項演算子)
- イテレータ

まとめ

プログラミングとは
なぜPythonを使うのか

プログラミング入門

Python文法(基礎)

Python文法(発展)

まとめ

プログラミングは機械に自分のやりたいことを伝える作業である

- 「人間が理解できる言語」から「機械が理解できる言葉」へ
- Pythonの文法に従って記述する

Pythonではデータ分析・機械学習ライブラリが利用できる

- Pythonのモジュール機能によって実現されている