プログラミング実習I クラス5 (井村担当)

知能・機械工学課程 井村 誠孝

m.imura@kwansei.ac.jp

for文による繰り返し処理

- for文を使うと，リストの各要素に対して，反復して処理を行うことができる．
- 書式

正確には，リストだけではなく，あらゆる反復可能な(iterable)オブジェクトに対して繰り返し処理を行うことが可能．

```
for 変数 in リスト:
    繰り返す処理(1)
    繰り返す処理(2)
          :
    繰り返す処理(n)
```

ここの処理を何度も行う

プログラムのある部分が繰り返し実行されるのは，これまでになかった特徴

繰り返される部分はどこか?

- 字下げ(インデント)されて書かれたコードが実行される.

```
list_tohoku = [5349, 5478, 5344, 4644, 4968, 6259]
for val in list_tohoku:
    print(val)
```

インデント

- 複数行が続けて字下げされている部分をブロックと呼び，ブロック全体が繰り返し処理の対象となる.

```
list_tohoku = [5349, 5478, 5344, 4644, 4968, 6259]
sum_tohoku = 0
for val in list_tohoku:
    print(val)
    sum_tohoku += val
```

ここがブロック

すごく重要: **Python**はインデントでブロックを作る

実行の様子を細かく追う

繰り返し 1回目

まず，変数 val の値が，リストの先頭の要素(5349)になり，ブロックの中の実行に移る．

```
for val in [5349, 5478, 5344, 4644, 4968, 6259]:
    print(val)                                    繰り返すブロック
```

5349

ブロックの中が繰り返される．この例ではブロックは1行のみ．
変数 val の値を出力する．valは5349なので，この値が出力される．

画面出力

```
5349
```

繰り返し 2回目

次に，変数 val の値が，リストの2番目の要素(5478)になり，ブロックの中の実行に移る．

```
for val in [5349, 5478, 5344, 4644, 4968, 6259]:
    print(val)                                    繰り返すブロック
```

5478

ブロックの中が繰り返される．この例ではブロックは1行のみ．
変数 val の値を出力する．valは5478なので，この値が出力される．

画面出力

```
5349
5478
```

関数 range() を使って連番を発生させる

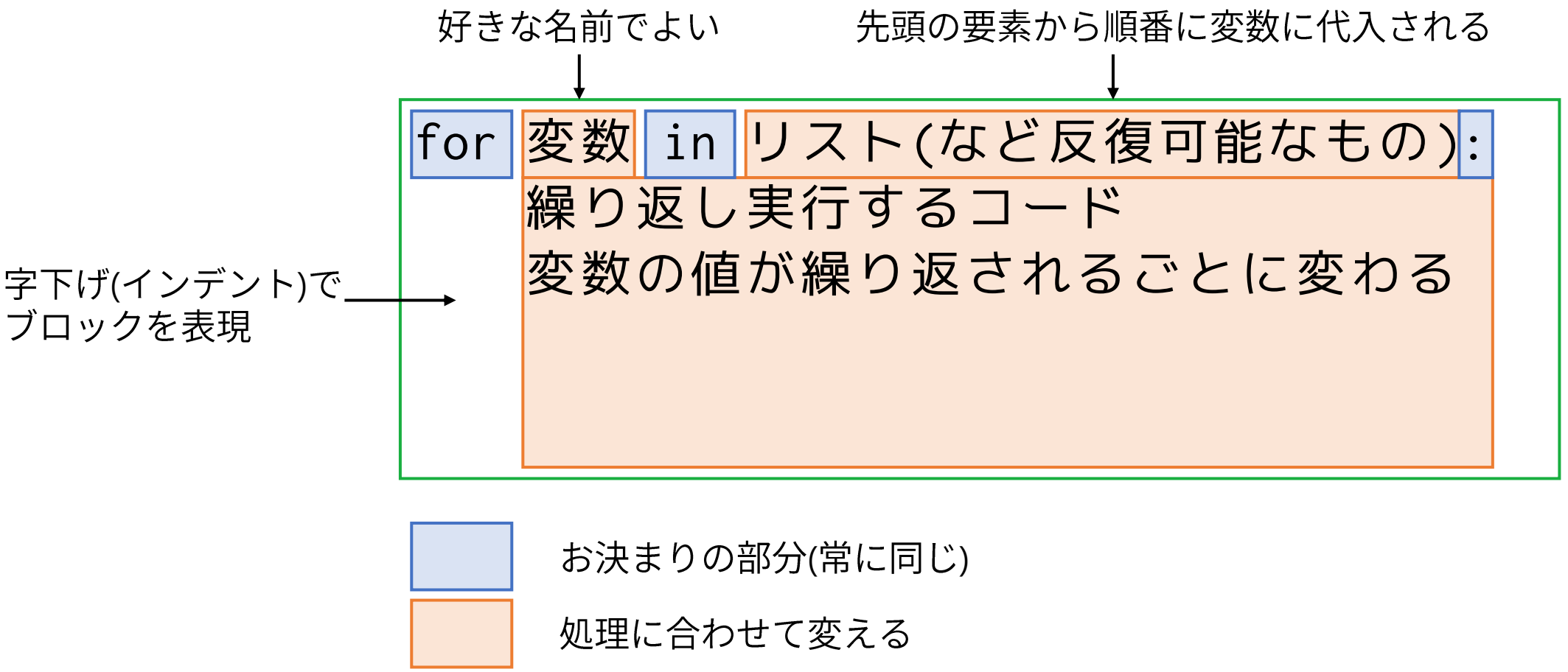
- range(5) → [0, 1, 2, 3, 4]と同じ効果
  - インタラクティブシェルで試してみる

半角スペースで
インデントする

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

  - 試してみよう
    - 1から100まで → for i in range(1, 101):
    - 0 から2つおきに10まで → for i in range(0, 11, 2):

for文の構造 まとめ


好きな名前でよい
先頭の要素から順番に変数に代入される
for 変数 in リスト(など反復可能なもの):
繰り返し実行するコード
変数の値が繰り返されるごとに変わる
字下げ(インデント)で
ブロックを表現
お決まりの部分(常に同じ)
処理に合わせて変える

if文による条件判断処理

- if文を使うと，ソースコードのある部分を，実行したりしなかったりすることができる．
  - TrueとFalseからなる真偽型を返す比較演算子(教科書 p.51)を利用して，条件によって処理を分ける．
  - 「もし，条件が成立すれば，…を行う．成立しなければ，…を行う．」
- 単純な例

注意!! 等しいかどうかの判定は == (イコール2つ)

条件は True → 実行される →

```
x = 0
if x == 0:
    print('0です')
else:
    print('0ではありません')
```

条件は False → 実行される →

```
x = 1
if x == 0:
    print('0です')
else:
    print('0ではありません')
```

if文のいろいろ

条件が成立したときのみ

```
if 条件式:
    条件成立時の処理
```

条件成立と不成立

```
if 条件式:
    条件成立時の処理
else:
    条件不成立時の処理
```

複数の条件

```
if 条件式1:
    条件1成立時の処理
elif 条件式2:
    条件2成立時の処理
elif 条件式3:
    条件3成立時の処理


else:
    全条件不成立時の処理
```

基本はこれで，elif節やelse節を省略すると
左の2つになる．

曜日を確認する例

条件が成立したときのみ

```
import datetime
today = datetime.datetime.now()

if today.weekday() < 5:
    print('頑張って働こう!')
```

条件成立と不成立

```
import datetime
today = datetime.datetime.now()

if today.weekday() < 5:
    print('頑張って働こう!')
else:
    print('休日だー')
```

複数の条件

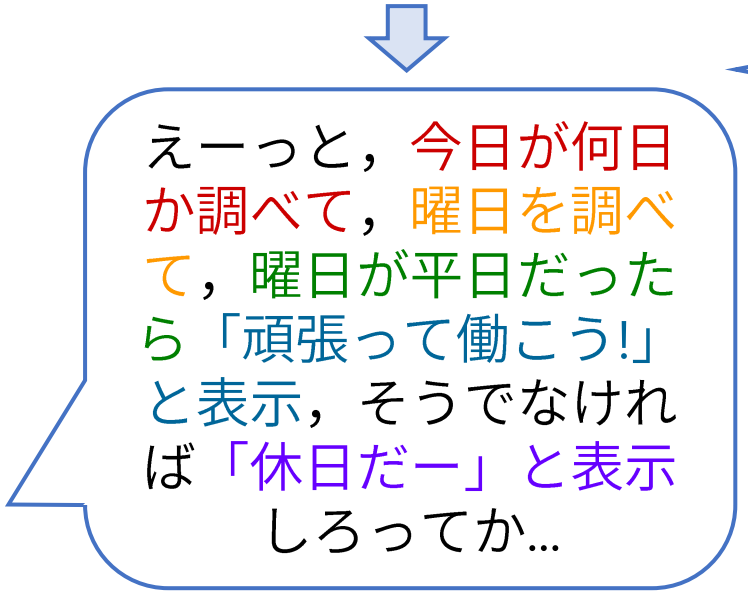
```
import datetime
today = datetime.datetime.now()

if today.weekday() < 4:
    print('頑張って働こう!')
elif today.weekday() == 4:
    print('ゆっくりやろう!')
else:
    print('休日だー')
```

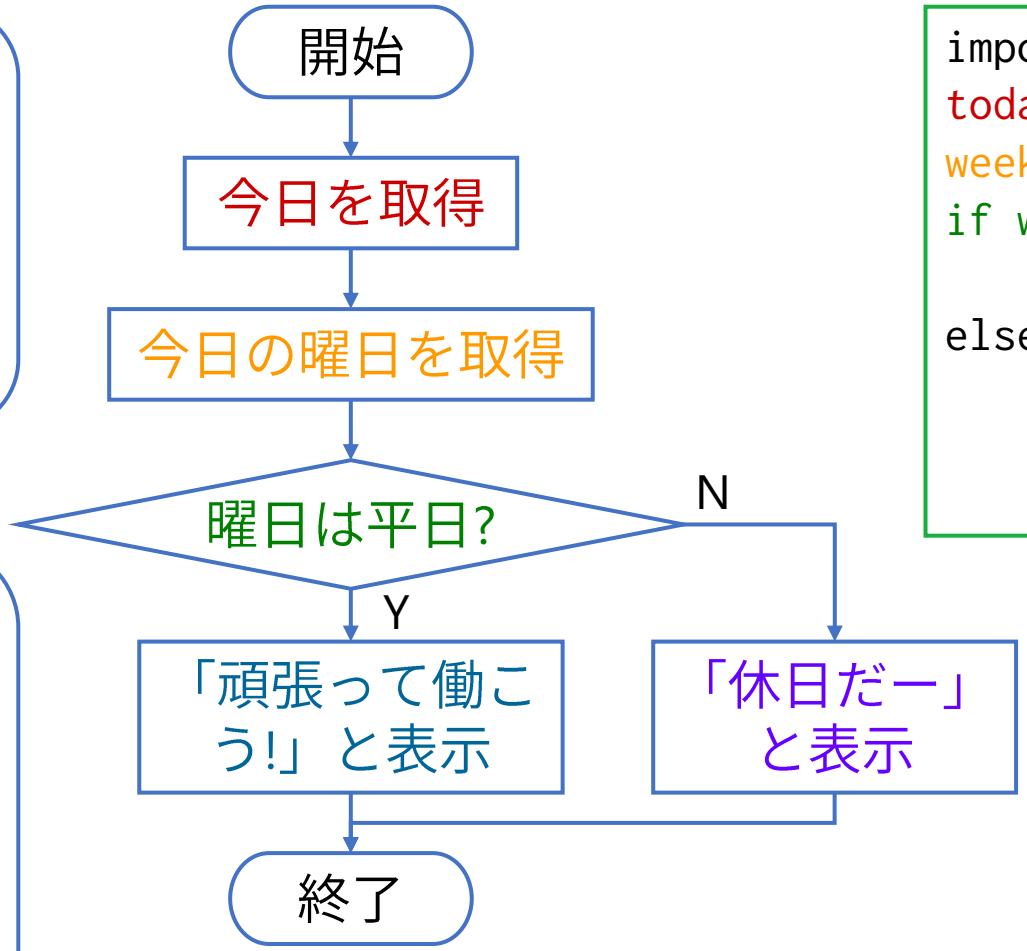
※ datetime型のメソッドweekday()の返値と曜日との対応は，
以前に登場したので各自確認してください．

処理の可視化: フローチャート

えーっと，今日がが平日だったら「頑張って働こう!」と表示，そうでなければ「休日だー」と表示しろってか…



えーっと，今日が何日か調べて，曜日を調べて，曜日が平日だったら「頑張って働こう!」と表示，そうでなければ「休日だー」と表示しろってか…




```
import datetime
today = datetime.datetime.now()
weekday = today.weekday()
if weekday < 5:
    print('頑張って働こう!')
else:
    print('休日だー')
```

教科書に無いが，重要

複雑な条件を記述する

- 2つの条件が共に成立 - and
- 例

```
x = 1
y = 2
if x > 0 and y > 0:
    print('xもyも正')
```

```
x = 10
if 5 <= x and x < 15:
    print('xは5以上15未満')
```

Pythonの場合は以下のように書くこともできる
(このように記述できない言語もある)

```
x = 10
if 5 <= x < 15:
    print('xは5以上15未満')
```

- 2つの条件の少なくとも一方が成立 - or
- 例

```
x = 1
y = -2
if x > 0 or y > 0:
    print('xかyのいずれかは正')
```

両方成立してもOK

```
x = 1
y = 2
if x > 0 or y > 0:
    print('xかyのいずれかは正')
```

orは一緒にできない

```
x = 20
if x <= 5 or 15 < x:
    print('xは5以下か15より大きい')
```

AND, OR, NOT

論理積(AND)

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

論理和(OR)

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

論理否定(NOT)

A	NOT A
0	1
1	0

参考: 排他的論理和(XOR)

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0