データ科学基礎演習B（3）

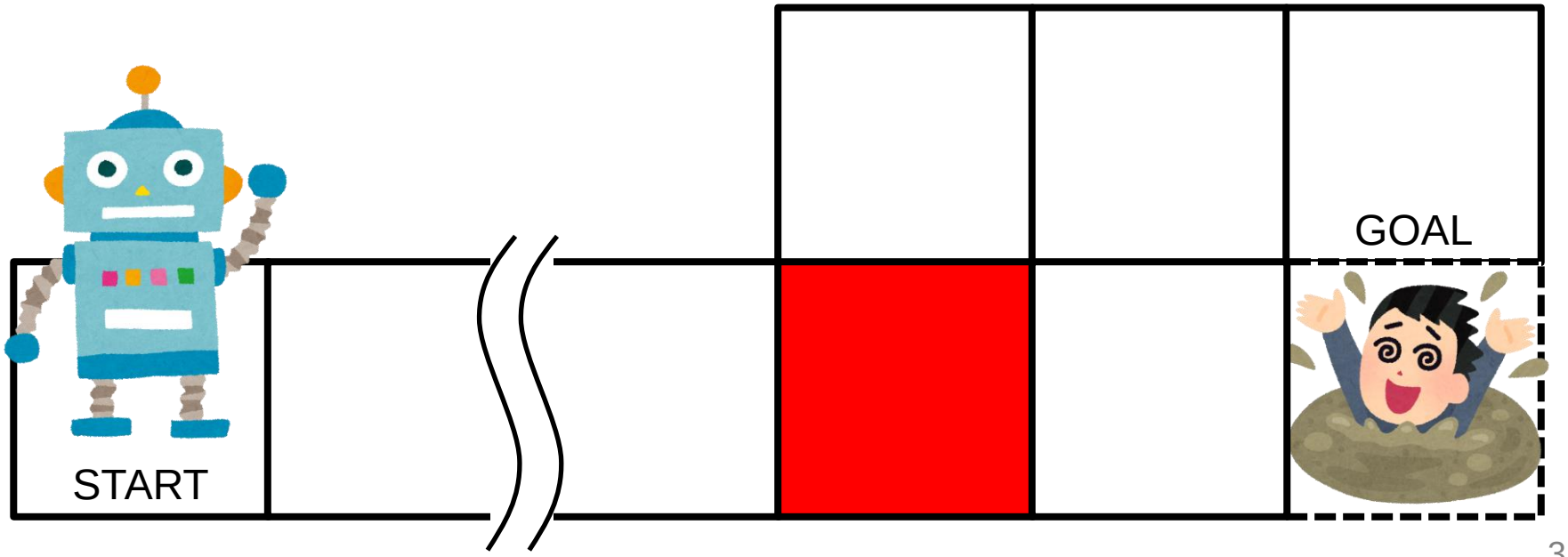
データ科学科目部会

制御構造（その2）

条件分岐（if, elif, else）をマスターしよう

どうすればロボットは目的地に着けるでしょう？

- 利用できる命令セット
  - ① １歩前に進む
  - ② X回繰り返す



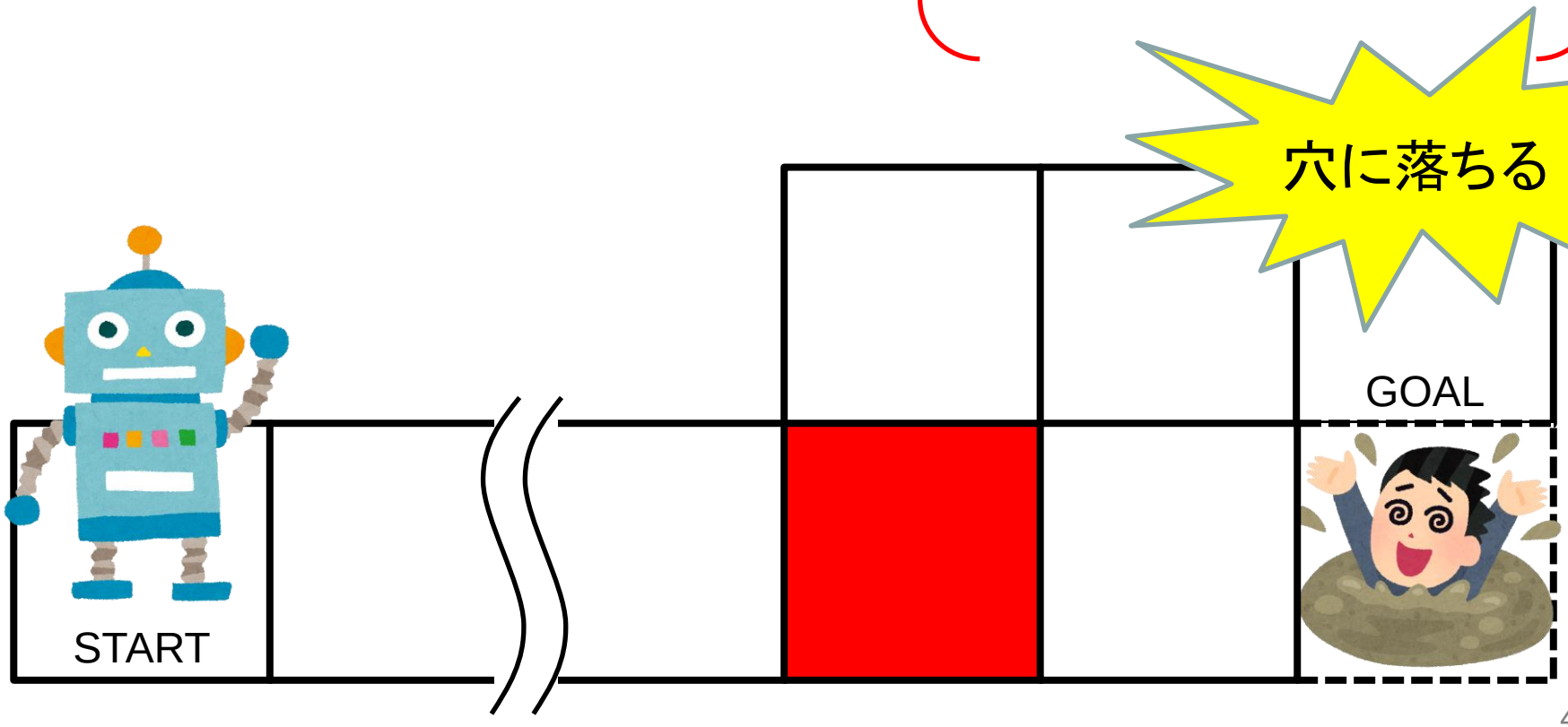
どうすればロボットは目的地に着けるでしょう？

- 利用できる命令セット
  - ① １歩前に進む
  - ② X回繰り返す

プログラム2

５回繰り返す

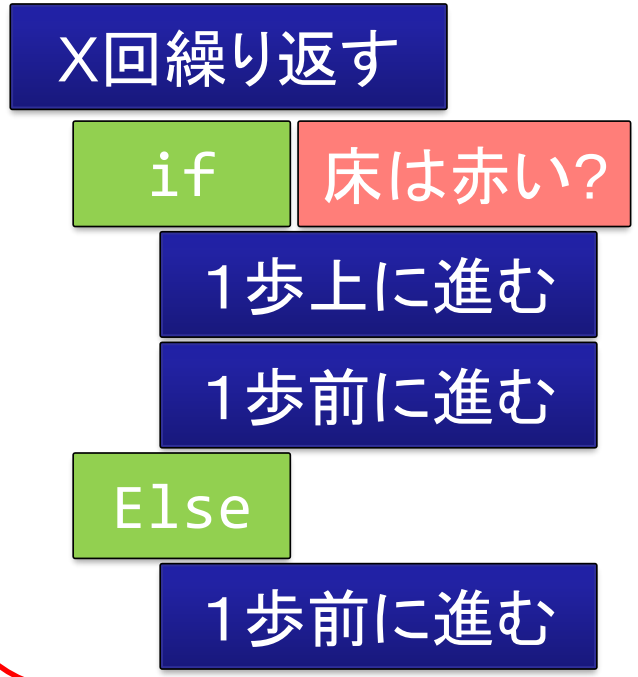
１歩前に進む

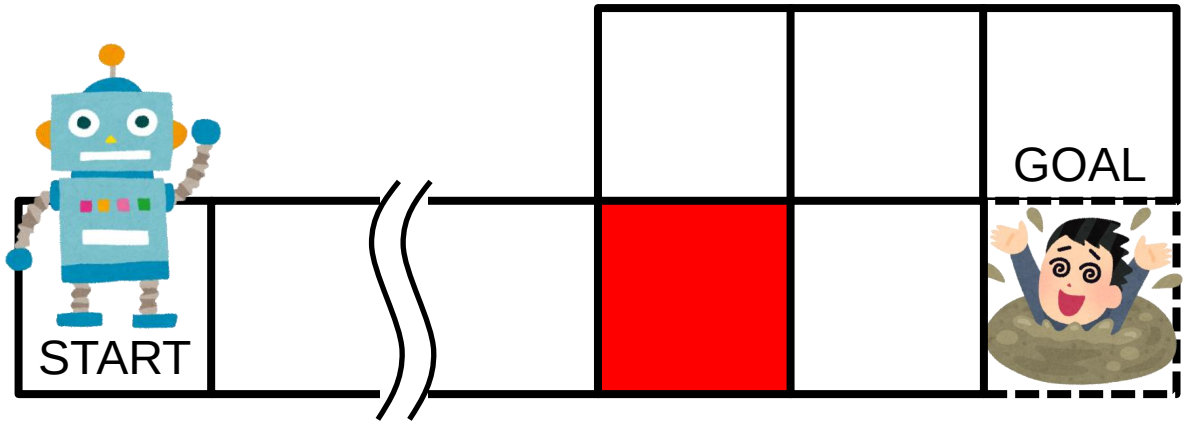


どうすればロボットは目的地に着けるでしょう？

- 利用できる命令セット
  - ① 1歩前に進む
  - ② 1歩上に進む
  - ③ X回繰り返す
  - ④ 条件分岐（if-else）
    - ① True なら命令1
    - ② False なら命令2
  - ⑤ 床は赤い?

プログラム2





条件分岐とは？

- 条件式の値に応じて処理を切り替えるしくみ

- Pythonで用意されている条件分岐は3種類
  - `if`
    - 最初の条件分岐（必ず最初は if から始まる）
  - `elif`
    - `if` を満たさなかった場合の条件分岐（複数可，省略可）
    - `else if` の略
  - `else`
    - `if` と `elif` のいずれも満たさなかった場合（省略可）

if-elif-else の構文

- 条件式は上から順に評価し，最初に満たしたものの処理ブロックが実行される

```
if 条件式1:
    …処理ブロック1…
elif 条件式2:
    …処理ブロック2…
elif 条件式3:
    …処理ブロック3…
else:
    …処理ブロック4…
```

条件式のあとにコロン(:)

省略可能

if-elif-else の構文

- 条件式は上から順に評価し，最初に満たしたものの処理ブロックが実行される
- ifのみ，ifとelifのみ，ifとelseのみ，も可能

```
if 条件式1:
    …処理ブロック1…
```

```
if 条件式1:
    …処理ブロック1…
elif 条件式2:
    …処理ブロック2…
```

```
if 条件式1:
    …処理ブロック1…
else:
    …処理ブロック2…
```

条件式の書き方

- 式の計算結果が真(True)もしくは偽(False)となるもの
  - 値の比較(大小関係，等号，不等号，...)
  - True, False そのものも条件式になる
    - True は常に真(条件を満たす) ← 無限ループに使える
    - False は常に偽(条件を満たさない)
  - 変数が左側に来なくてもいい

```
while True:
    …処理ブロック…
```

- and，or，not を組み合わせると複雑な条件が作れる

- 複雑な条件式の例

①変数 i が 0 以上かつ10未満

```
0 <= i and i < 10
i >= 0 and i < 10
```

同じ

②変数 i が 0 ではない

```
not i == 0
i != 0
```

同じ

条件式の優先順位(1)

- 次の条件式はどう解釈すれば良いでしょうか？

```
i < 0 or 10 < i and not i > 20
```

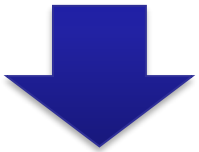
not → and → or
の順に条件式が評価される

条件式の優先順位(2)

- 条件式を組み合わせる場合はカッコでくくろう

```
i < 0 or 10 < i and not i > 20
```

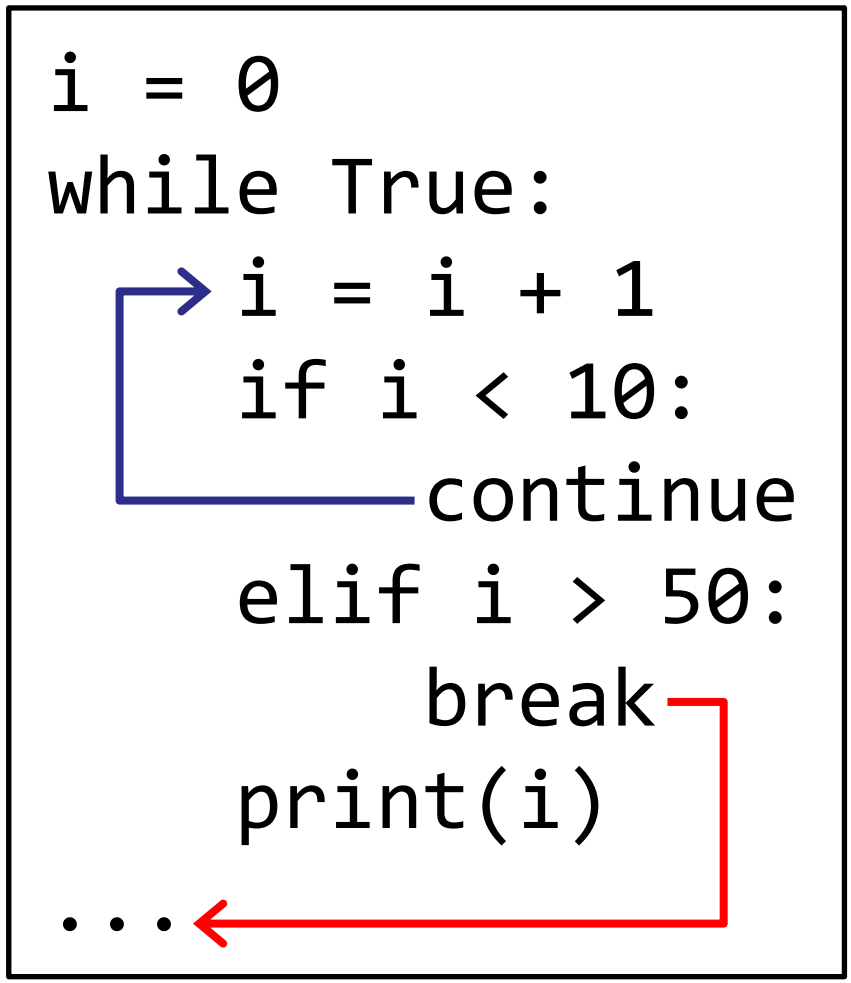
```
i < 0 or (10 < i and not i > 20)
```



```
i < 0 or (10 < i and i <= 20)
```

条件分岐を使った繰り返しの途中終了

- forやwhileの途中で繰り返しを終了する方法
  - break
    - 実行中のforやwhileの繰り返しを終了する
  - continue
    - 処理をスキップしてforやwhileの先頭に戻る



99以下の素数を求めてみよう

- 素数：1より大きい自然数で1と自分自身以外では割り切れない数

```
for i in range(2,100):
    sosu = True
    for j in range(2,i):
        if i % j == 0:
            sosu = False
            break
    if sosu:
        print(i)
```

データ集合

配列（リスト），タプル，辞書をマスターしよう

配列 or リスト

- データを並べたデータ構造（角括弧を使う）

  例1）`[3, 1, 1, 4, 3, …]`

  例2）`['If', 'I', 'am', 'a', 'bird']`

- Pythonで利用できる配列の種類
  - list　← 組み込み型（何もせず利用可能）
    - 異なる型（整数，文字列，他）を一緒に格納できる
  - array　← 標準ライブラリ array をインポート
    - 同じ型しか格納できないが list より効率が良い
  - numpy.ndarray　← 後日紹介
    - 同じ型しか格納できないが非常に高速
    - 数値計算等の様々な関数が利用可能

配列(list)の作り方(1)

- 空っぽの配列を作成

```
a = []
```

- 中身を指定して作成

```
a = [3, 1, 3, 4]
```

要素はカンマ(,)で区切る

- 同じ要素を指定個数並べて作成

```
a = [0]*4
```

※ [0,0,0,0] になる

- 内包表記を使って [0,1,2,3,4] を作成

```
a = [i for i in range(5)]
```

for のループ変数 i を使って
配列の要素を一つずつ挿入

配列(list)の作り方(2)

```
[1]  1  a = []
     2  a
```

```
[]
```

```
[2]  1  a = [3,1,3,4]
     2  a
```

```
[3, 1, 3, 4]
```

```
[3]  1  a = [1]*5
     2  a
```

```
[1, 1, 1, 1, 1]
```

```
[4]  1  a = [i for i in range(1,6)]
     2  a
```

```
[1, 2, 3, 4, 5]
```

```
     1
```

配列(list)の要素へのアクセスと要素数

- 要素数N個の配列のインデックスは 0 ～ N-1

- 角括弧にインデックスを指定して要素にアクセス
  - 0 番目の要素に値を代入

    ```
    a[0] = 10
    ```

  - 1番目の要素の値を取得

    ```
    b = a[1]
    ```

- 配列(list)の要素数: len(a)

```
[1]  1  a = [1, 2, 3, 4, 5]
     2  a

     [1, 2, 3, 4, 5]

[2]  1  a[0] = 10
     2  a

     [10, 2, 3, 4, 5]

[3]  1  a[1]

     2

[4]  1  len(a)

     5
```

配列(list)に対する操作方法

- 配列に対して何か操作をする場合は変数名にドットをつけて処理方法(関数名)を指定する

配列の変数 . (ドット) 関数名

append や pop など

配列(list)の要素の追加

- 末尾に要素を追加
  - a.append(値)
- 末尾に配列を追加
  - a.extend(配列)
- 指定した位置に挿入
  - a.insert(挿入位置, 値)

```
[1]  1  a = []
     2  a
```

```
[]
```

```
[2]  1  a.append(1)
     2  a
```

```
[1]
```

```
[3]  1  a.extend([2,3])
     2  a
```

```
[1, 2, 3]
```

```
[4]  1  a.insert(0,99)
     2  a
```

```
[99, 1, 2, 3]
```

配列（list）の要素の削除

- 指定した位置の要素を削除
  - a.pop(削除位置)

- 指定した値を検索して最初に見つかったものを削除
  - a.remove(削除する値)

```
[1]  1  a = [1, 2, 3, 3, 3, 4, 5]
     2  a
```

```
[1, 2, 3, 3, 3, 4, 5]
```

```
[2]  1  a.pop(0)
     2  a
```

```
[2, 3, 3, 3, 4, 5]
```

```
[3]  1  a.remove(3)
     2  a
```

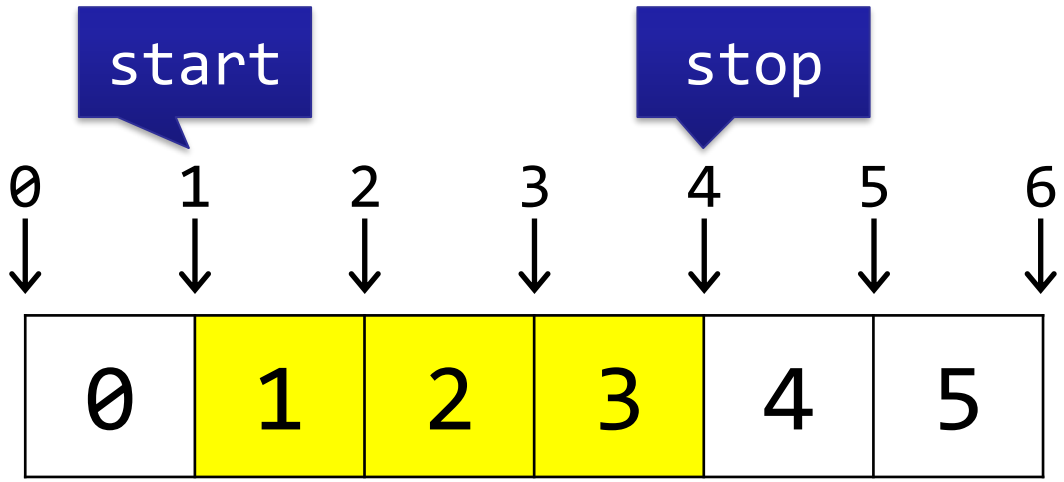
```
[2, 3, 3, 4, 5]
```

配列(list)のスライス(1)

- 配列の一部を切り取る方法
  - start から stop で囲まれる範囲を

配列の変数[start:stop]

例）a[1:4] の場合



配列(list)のスライス(2)

- start と stop は省略できる
  - start を省略すると
    先頭(0)を指定
  - stop を省略すると
    末尾(len(a))を指定

```
[4]  1  a = [0, 1, 2, 3, 4, 5]
     2  a
```

```
[0, 1, 2, 3, 4, 5]
```

```
[5]  1  a[1:4]
```

```
[1, 2, 3]
```

```
[6]  1  a[:4]
```

```
[0, 1, 2, 3]
```

```
[7]  1  a[1:]
```

```
[1, 2, 3, 4, 5]
```

配列(list)の要素の表示(1)

- インデックスを利用して配列(list)の要素を表示

```
for i in range(N):
    …処理…
```

ここに配列の要素数を指定

```
[1]  1  arr = [1, 5, 9, 3]

[2]  1  for i in range(len(arr)):
     2  |  print(arr[i])

     1
     5
     9
     3
```

配列(list)の要素の表示(2)

- 配列(list)の変数をデータ集合として利用

```
for 変数 in データ集合:
    …処理…
```

ここに配列を指定

```
[1]  1  arr = [1, 5, 9, 3]

[2]  1  for a in arr:
     2  |  print(a)

     1
     5
     9
     3
```

配列(list)の使い方練習

① 1 ～ 9 を要素に持つ配列を作成してみよう

② ①の配列の末尾に 10 を追加してみよう

③ ②の配列の先頭の要素を削除してみよう

④ ③の配列の 2 ～ 4 番目の要素をスライスの機能を使って切り出してみよう

タプル（tuple）

- データを並べたデータ構造（丸括弧を使う）

  例1）(3, 1, 1, 4, 3, …)

  例2）('If', 'I', 'am', 'a', 'bird')

- 配列との違い
  - タプルは丸括弧（配列は角括弧）
    - タプルでは丸括弧を省略できる
  - 要素の追加や削除はできない
  - 複数の変数をまとめて管理する場合に活用する

タプル（tuple）の使い方

- 基本的に中身を指定して作成

```
a = (3, 1, 3, 4)
a = 3, 1, 3, 4
```

どちらで書いても同じ！

- 要素数N個のタプルのインデックスは 0 ～ N-1

- 角括弧にインデックスを指定して要素にアクセス
  - 1番目の要素の値を取得 ※要素への代入はできない

```
b = a[0]
```

- タプル（tuple）の要素数: len(a)

辞書（dict）

- キーと値のペアでデータを管理するデータ構造

キー	:（コロン）	値

- 辞書（dict）の例

  例1）`{0:1, 1:2, 8:9}`

  例2）`{'I':3, 'am':8, 'a':22, 'bird':0)`

- 配列との違い
  - 辞書は波括弧を使う（配列は角括弧）
  - キーを使って値を高速に検索できる
    - キーは好きなものを指定できる（配列は0,1,2,…）

辞書(dict)の使い方(1)

- 空っぽの辞書を作成

```
a = {}
```

- 中身を指定して作成

```
a = {0:1, 1:2, 8:9}
```

- 角括弧にキーを指定して要素にアクセス
  - 'key1' というキーに値を代入(自動で要素を追加)

```
a['key1'] = 10
```

  - 'hoge' というキーの値を取得

```
b = a['hoge']
```

- 辞書(dict)の要素数: `len(a)`

辞書(dict)の使い方(2)

```
[1]  1  a = {}
     2  a
```

```
{}
```

```
[2]  1  a = {0:1, 1:2, 9:99}
     2  a
```

```
{0: 1, 1: 2, 9: 99}
```

```
[3]  1  a['key1'] = 10
     2  a
```

```
{0: 1, 1: 2, 9: 99, 'key1': 10}
```

```
[4]  1  a['key1']
```

```
10
```

```
[5]  1  len(a)
```

```
4
```

```
     1
```

辞書(dict)の要素を表示

- items関数を使って辞書の中身を取得
  – ヒント）items関数はキーと値のペアをタプルで返す

```
[1]  1  a = {'key1':0, 'key2':3, 'key3':99}
     2  a
```

```
{'key1': 0, 'key2': 3, 'key3': 99}
```

```
[2]  1  for (k,v) in a.items():
     2  |  print('{0} - {1}'.format(k,v))
```

```
key1 - 0
key2 - 3
key3 - 99
```

配列・タプル・辞書の違い（まとめ）

	配列	タプル	辞書
インデックス	`0,1,…,N-1`	`0,1,…,N-1`	任意
値の変更	○	×	○
追加	○	×	○
削除	○	×	○
スライス	○	○	×
括弧の省略	×	○	×
要素数		`len(a)`	

配列と辞書の入れ子構造

- 配列（list）や辞書（dict）は互いを自身の要素とする入れ子構造を作ることができます
  - 例1）配列の要素が辞書

```
a = [{'key1':0, 'key2':2},
     {'key1':9, 'key2':3}]
```

※ a[0] は {'key1':0, 'key2':2}

  - 例2）辞書の要素が配列

```
a = {'key1':[0,1,2,3],
     'key2':[3,8,0,2,5]}
```

※ a['key1'] は [0,1,2,3]

入れ子構造のデータアクセス

- 角括弧で順に要素を指定してアクセス

```
[1]  1  a = [{'key1':0, 'key2':2},
     2       {'key1':9, 'key2':3}]
     3  print(a)
```

```
[{'key1': 0, 'key2': 2}, {'key1': 9, 'key2': 3}]
```

```
[2]  1  a[0]
```

```
{'key1': 0, 'key2': 2}
```

```
[3]  1  a[0]['key2']
```

```
2
```

課題

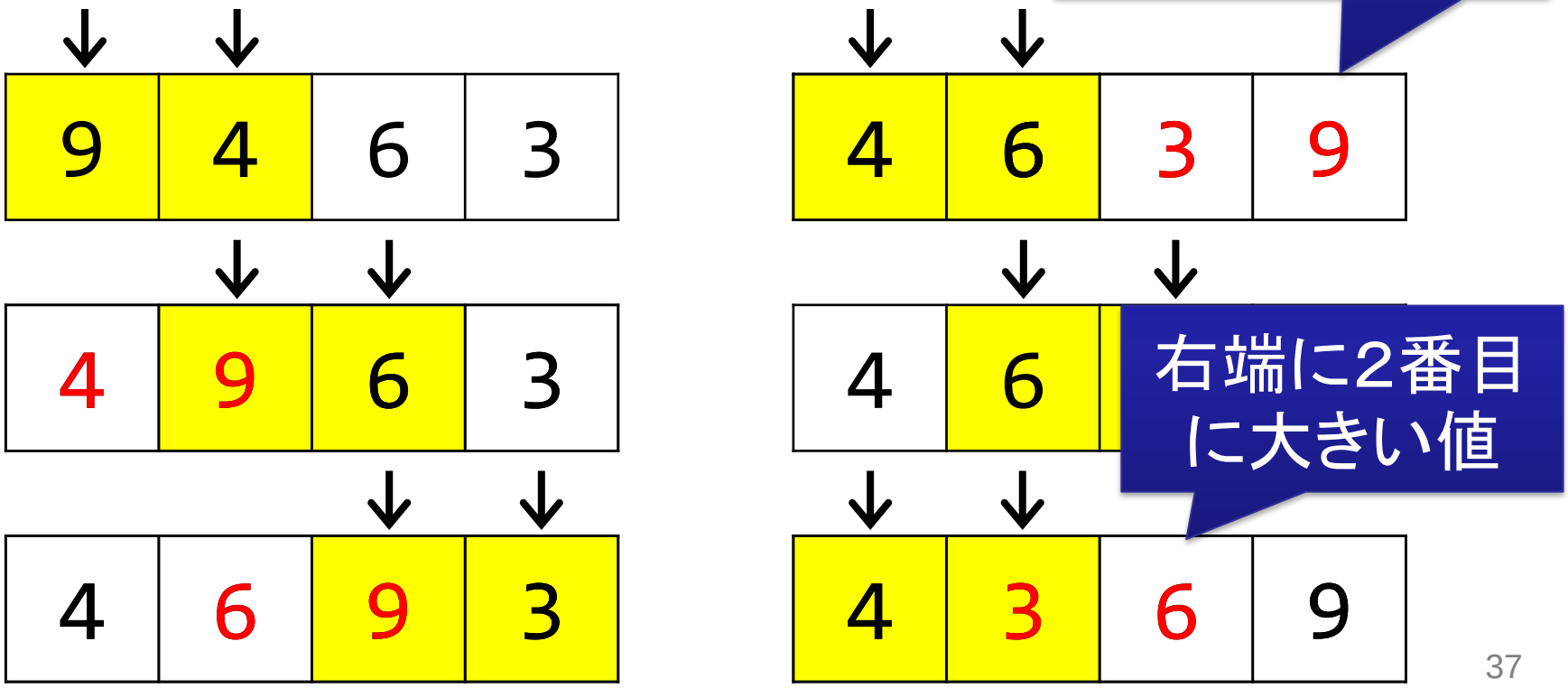
- 配列の要素を順番に並べ替えてみよう

- 下記URLから演習課題のノートブックを自身のGoogle Driveにコピーし，Google Collaboratoryを起動して各設問に回答すること
  - https://bit.ly/3gKE09B

- 演習課題を提出する際は，ノートブックのURLで共有し，そのURLを提出すること
  - 課題のヒントは次ページで紹介

課題のヒント（バブルソート）

- データを昇順／降順に並び替えるアルゴリズム
  - 隣り合うデータの順序が逆であれば入替える処理の反復
- バブルソートの仕組み



課題のヒント（バブルソート）

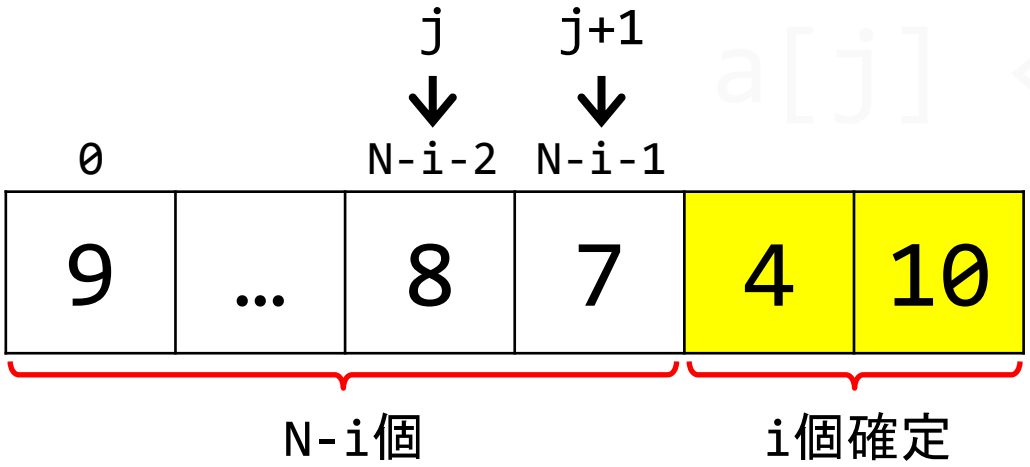
- バブルソートの疑似アルゴリズム
  - ソート対象のは要素数Nの配列a

```
for i = 0,1,2,…,N-2
    for j = 0,1,2,…,N-i-2
        if a[j] > a[j+1] then
            a[j] <-> a[j+1]
```

課題のヒント（バブルソート）

- バブルソートの疑似アルゴリズム
  - ソート対象のは要素数Nの配列a

```
for i = 0,1,2,…,N-2
    for j = 0,1,2,…,N-i-2
        if a[j] > a[j+1] then
```



課題のヒント（バブルソート）

- バブルソートの疑似アルゴリズム
  - ソート対象のは要素数Nの配列a

```
for i = 0,1,2,…,N-2
    for j = 0,1,2,…,N-i-2
        if a[j] > a[j+1] then
            a[j] <-> a[j+1]
```

変数の値の交換

課題のヒント（バブルソート）

- 変数の値の交換方法
  - ダメなやり方

```
a[j]   = a[j+1]
a[j+1] = a[j]
```

a[j]=a[j+1]をした際にa[j]の値が失われる

  - 正しいやり方

```
tmp    = a[j]
a[j]   = a[j+1]
a[j+1] = tmp
```

a[j]に値を代入する前に一時保存する

課題提出時の注意点

- 課題提出の際には提出前に必ず以下2点を確認すること
  - Google Collaboratoryの共有設定の際には『リンクを知っている全員』にチェックを入れる
  - 課題提出時に貼り付けたURLの末尾が「?usp=sharing」となっている（共有設定を開き、「リンクのコピー」ボタンを押して共有用のURLをコピー＆ペーストする）