4. 条件分岐と繰り返し 2

プログラミング・データサイエンス I

2024/5/2

1　今日の目的

今日の目的

- while は無限ループになる危険性がある
- 予め繰り返し回数が定まっていることが多い
  - `for` 文を使う
- 予めエラーが予見できるときの処理

前回は、条件を満たすまで繰り返す `while` という構文を説明しました。`while` は、予め繰り返し回数を指定できません。そのため、無限ループとなる危険性があることも注意しました。

実際のプログラムを書く際には、予めその繰り返し回数が定まっている場合が、非常に多くあります。例えば、データを全て足す場合には、データの個数が繰り返し回数になります。このように、予め繰り返し回数が分かっている場合には、`for` を使います。`for` を使った繰り返しブロックを `for` ループとも言います。前回、`while` ループを *indefinite* (無限定) ループと言いました。`for` ループは、*definite* (限定) ループとも言います。

また、プログラムが正しくても、エラーが発生する場合があります。エラーが予見できる場合に、プログラムと対策を講じる方法についても説明します。

2　for を使った繰り返し: for loops (definite iterations)

2.1　for を使った繰り返しの基本

for を使った繰り返しの基本

- 指定回数繰り返す
- for 変数 in 範囲
- for x in range（n）
- x を 0 から n-1 まで変化させながら繰り返す

for 文の基本的構文は、「for 変数 in 範囲」です。変数が整数の場合には、その範囲を range() で指定します。この時、範囲を表す range(n) は、0 から n-1 までの整数を表します。n を含まないことに注意してください。for ループの処理イメージを図 1 に示します。

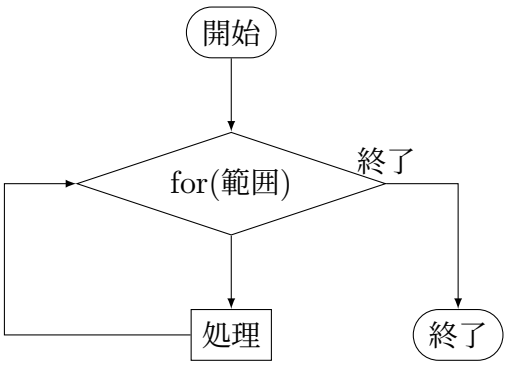

図 1　for ループによる繰り返し

前回配布したサンプルプログラムのうち、for.ipynb を開けてください。最初のセルの例 (ソースコード 2.1) で、i という変数を 0 から 10 まで変化させて、sum1 に加えています。答えは 55 になりましたか? for ループで、整数を順に変えながら操作するとき、その変数をループインデクス (loop index) と言います。通常は、i、j、k などを使います。

ソースコード 2.1　for の基本

```
sum1 = 0
for i in range(11):
    sum1 += i
print(sum1)
```

2.2　range の使い方

range の使い方

- 繰り返しの範囲を指定する
- range(開始, 終了, ステップ)

range() には、値を 3 個指定することもできます。開始、終了、一回あたりの変化量を指定できます (ソースコード 2.2)。一回あたりの変化量を指定しない場合には、つまり、値を 2 個だけ指定すると、一回あたり 1 だけ変化します。

ソースコード 2.2　range の利用

```
for i in range(3, 12, 3):
    print(i)
```

課題 2.1　for ループのインデクスは、下げていくことも可能です。k の値を 10 から 1 まで、一つづつ下げていくことで、その和を計算しなさい。

2.3　range を使わない繰り返し

range を使わない繰り返し

- リストなどの要素について繰り返す例

繰り返し範囲は、整数のようなものである必要はありません。iterable、つまり順番に辿ることができるものを指定することができます。ソースコード 2.3 では、colors というリストの各要素に対して、プリントをしています。リストは、要素を一列に並べた物

で、先頭から順に要素を辿ることができます。まだ説明していませんから、「だいたいこんな感じ」として見ておいてください。

ソースコード 2.3　リストの要素に対する繰り返し

```
colors = ["red", "green", "orange", "blue"]#文字列のリスト
for c in colors:
    print(c)
print("-------")
```

リストは、その要素を一列に並べて管理し、要素には先頭から 0、1、2 と番号 (index) が付いています。それを使っているのが、ソースコード 2.4 の例です。`len()` という関数は、引数、つまり `()` の中の変数について、その長さを測ります。`len(colors)` は、`colors` というリストの要素数を返します。これも、「だいたい」の理解で十分です。

ソースコード 2.4　リストの要素に対する繰り返し

```
for i in range(len(colors)):
    print(colors[i])
print("-------")
```

2.4　`for` の入れ子 (nesting)

`for` の入れ子 (nesting)

- 多重ループ
- `for` 文の処理の中に `for` 文を書く

ネスティング (nesting) というのは「入れ子」、つまり、ある構造のなかに、同様の構造が入っていることです。ここでは、`for` による繰り返しの中に、`for` による繰り返しが入っている例を見ていきます。

ソースコード 2.5 を見てください。外側の `for` では、`i` を 0 から 2 まで変化させ、内側の `for` では、`j` を 0 から 3 まで変化させます。実行すると、すぐわかりますね。ソースコードを読む上では、インデントの深さに注意してください。

ソースコード 2.6 では、内側の `j` を使った `for` ループの上限は、外側の `for` ループのインデクス `i` になっています。つまり、内側の `for` ループの長さが、外側の `for` ループ

ソースコード 2.5　ネスティングした for

```
for i in range(3):
    print (f'i={i}')
    for j in range(4):
        print(f'({i}, {j})')
```

ソースコード 2.6　ネスティングした for

```
n = 5
for i in range(n, 0, -1):
    for j in range(i + 1):
        print(f'({i}, {j})')
```

インデクス i に応じて、変化してきます。動作を確認しましょう。

少し変わった例も示しましょう。de Morgan の法則は、論理演算の重要な定理の一つです。¬ を not (否定)、∨ を or (論理和)、∧ を and (論理積) と表すと、de Morgan の法則は、

$$\neg(p \vee q) = (\neg p) \wedge (\neg q) \tag{2.1}$$
$$\neg(p \wedge q) = (\neg p) \vee (\neg q) \tag{2.2}$$

と表すことができます。これを真理値表を作ることで確かめる例をソースコード 2.7 に示します。実行してみると、r と x、s と y が同じ論理値、つまり de Morgan の法則が成り立つことが解ります。

ソースコード 2.7　de Morgan の法則

```
for p in [True, False]:
    for q in [True, False]:
        r = (not (p or q))
        s = (not (p and q))
        x = ((not p) and (not q))
        y = ((not p) or (not q))
        print(f'{r} {x} {s} {y}')
```

3 break と continue: break and continue in for loops

break と continue

- break: for ループから出る
- continue: それ以降の処理を飛ばして for ループの先頭へ
  - for のカウントが一つ増えることに注意

while の際にも出てきた、break と continue を見ていきましょう。動作は、while の時とほぼ同じです。

break に至ると、for ループを抜け出します。continue は、for ループ内のそれ以降を実施せずに、for の最初に戻ります。while と異なる点として、for ループ内の continue の場合には、for で変化させている変数が、次の値になることに注意が必要です。

ソースコード 3.1 を見てください。リスト data の中から、負の要素を見つけると for ループから抜け出ます。偶数の要素を見つけると、次の要素に移ります。結局、正の奇数だけを out というリストに追加し、最後に印刷します。

ソースコード 3.1　break と continue

```
1  data=[2, 5, 7, 9, 11, -3, 8, -11, 10, 15]
2  out = []
3  for x in data:
4      if x < 0:
5          print('負の要素を発見。ループ中断')
6          break
7      if x%2 == 0:
8          continue
9      out.append(x)
10 print(out)
```

4 for-else 構文: for-else in for loops

for-else 構文

- for ループが break で終了しなかった場合の処理

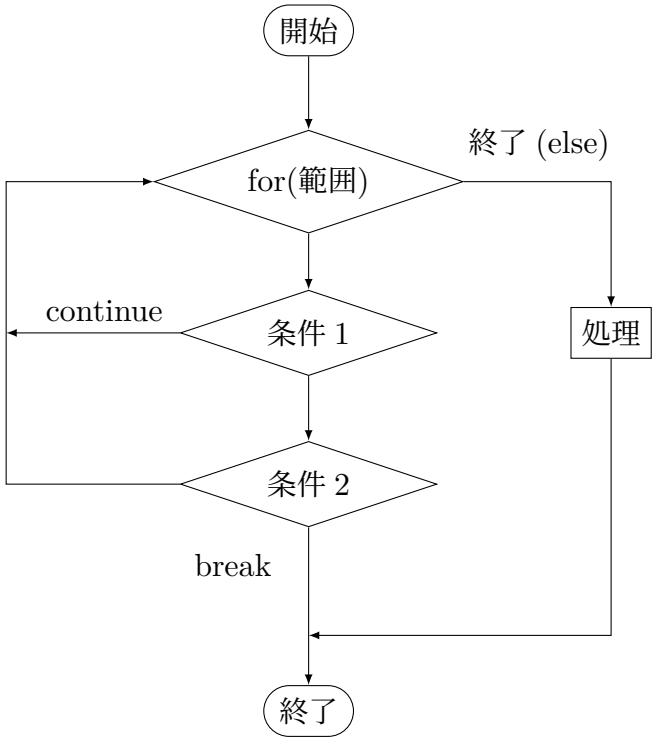

図 2　for ループにおける break、continue、そして else

for 文にも、対応する else があります。for ループの中で、break で終了しなかった場合に実行する部分です。

ソースコード 4.1 を見てください。ソースコード 3.1 とほぼ同じです。しかし、out を印刷するのは、break しない場合だけですから、例の data では、out を印刷しません。

ソースコード 4.1　for-else 構文

```
data=[2, 5, 7, 9, 11, -3, 8, -11, 10, 15]
out = []
for x in data:
    if x < 0:
        print(' 負の要素を発見。ループ中断')
        break
    if x%2 == 0:
        continue
    out.append(x)
else:
    print(out)
```

5 統計の例：dataSum0.ipynb as an example of statistics

5.1 すこしだけ数学

すこしだけ数学

- 和の記号
- 平均
- 標準偏差

ちょっと数学の話をしましょう。数式が出てきますが、行うことは、ほぼ四則演算だけですから、難しくありません。

データに名前を付けましょう。a、b、c と名前を付けるのではなくて、d_0、d_1、d_2 と右下に番号、インデクス (index) といいます、を付けていきましょう。ゼロから番号が始まるのは、コンピュータの分野の習慣です。d_i と書いて、i が 0 から $n-1$ まで n 種類あることにしましょう。Σ は、範囲を指定した和の記号です。Σ の下に書いているのが下限、上が上限です。

最も簡単な例を示します。式 (5.1) の左辺の記号は i を 0 から 4 まで加えることを表しています。

$$\sum_{i=0}^{4} i = 0 + 1 + 2 + 3 + 4 \tag{5.1}$$

また、i が 0 から 4 まで、5 種類あるデータの和は

$$\sum_{i=0}^{4} d_i = d_0 + d_1 + d_2 + d_3 + d_4 \tag{5.2}$$

と表します。

一般化しましょう。n 個のデータ d_i を考えます。

$$s = \sum_{i=0}^{n-1} d_i \tag{5.3}$$

$$s_2 = \sum_{i=0}^{n-1} d_i^2 \tag{5.4}$$

s がデータの和、s_2 がデータを二乗した和です。

ここから、データの平均 $\langle d \rangle$ と二乗の平均 $\left\langle d^2 \right\rangle$ を求め、分散 σ^2 を求めます。平均は、データの和をデータの個数で除したものです。分散は、データの平均からの差を二乗したものの平均です。

$$\langle d \rangle = \frac{1}{n} \sum_{i=0}^{n-1} d_i \tag{5.5}$$

$$\left\langle d^2 \right\rangle = \frac{1}{n} \sum_{i=0}^{n-1} d_i^2 \tag{5.6}$$

$$\begin{aligned} \sigma^2 &= \frac{1}{n} \sum_{i=0}^{n-1} \left(d_i - \langle d \rangle \right)^2 = \frac{1}{n} \sum_{i=0}^{n-1} \left(d_i^2 - 2 d_i \langle d \rangle + \langle d \rangle^2 \right) \\ &= \frac{1}{n} \sum_{i=0}^{n-1} d_i^2 - 2 \langle d \rangle \frac{1}{n} \sum_{i=0}^{n-1} d_i + \langle d \rangle^2 \frac{1}{n} \sum_{i=0}^{n-1} 1 \\ &= \left\langle d^2 \right\rangle - \langle d \rangle^2 \end{aligned} \tag{5.7}$$

分散の平方根である σ が標準偏差です。標準偏差は、平均の周囲にデータが広がる様子を表します。

5.2 python で計算する

python で計算する

- データの塊に対して和を求める
- 平均と標準偏差の計算
- dataSum0.ipynb

それでは、dataSum0.ipynb の中身を見てみましょう。最初のセルでは、data というリストを作っています (ソースコード 5.1)。リストについては、何回か先で説明します。なお、2 行目の len() という関数は、様々なモノの長さを計るときに出てきます。

ソースコード 5.1　データの定義

```
1 data = [80, 85, 90, 50, 100, 80, 75, 95, 100, 60, 80, 70, 65, 90, 85]
2 n = len(data)
3 print("データ数 " + str(n))
```

ソースコード 5.2 では、for を使って、データの和と、データの二乗の和を求めていま

す。data から一つ一つ取り出して、sum に加え、sum_of_square に二乗して加えているということを理解してください。最後にソースコード 5.3 では、データ及びその二乗の和をデータ総数で除するなどによって、データの平均と標準偏差を求め、印刷しています。7 行目と 8 行目の print() では、% の前が書式、後ろが印刷する変数名です。ここでは、小数点以下 2 桁で印刷するように指示しています。

ソースコード 5.2　データの和

```
1 sum = 0 #和を保存
2 sum_of_square = 0 #二乗和を保存
3 for d in data:
4     sum  += d
5     sum_of_square += d*d
```

ソースコード 5.3　平均と標準偏差

```
1 average = sum / n
2 average_of_square = sum_of_square / n
3 dispersion = average_of_square - average * average #分散
4 deviation = math.sqrt(dispersion) #標準偏差
5
6 #小数点以下 2 桁で出力
7 print("平均 %.2f" % average)
8 print("標準偏差 %.2f" % deviation)
```

6　例外処理：Exceptions

例外処理：Exceptions

- 実行中にエラーが発生すると、プログラムの実行は、そこで停止。
- 予期しないデータ、ディスクの状態、ネットワークの状態で、エラーが発生する可能性
- 予め、エラー発生を予測し、エラー発生時の対応を記述しておくことで、その後の処理を継続

最後に「例外処理」について説明します。ここも、少し難しいので、「必要になってから学びなおし」で十分です。

プログラムを実行している最中でエラーが発生すると、プログラムはその場所で止まってしまいます。どのようなエラーがあり得るかを少し考えましょう。

例えば、

1. 数値のデータをファイルから読み込もうとしているのに途中に数値でないモノが混じっている
2. ディスクからファイルを読もうとしたら、ファイルが無かった
3. インターネットからデータを取得しようとしたが、通信状態が悪くて失敗した
4. データによっては、ゼロでの割り算が発生してしまう

などというエラーを予想することができます。

プログラムによっては、エラーを予測し、エラーが発生した際には適切な処理 (例えば、エラーメッセージを出す) を行って、プログラム全体を止めずにおく必要があることがあります。これが例外処理です。

ソースコード 6.1　例外処理の基本

```
try:
    エラー発生可能性のある処理
except:
    エラー時の処理
```

ソースコード 6.2　例外の種類に応じた切り分け

```
try:
    エラー発生可能性のある処理
except 例外 1:
    例外 1 時の処理
except 例外 2:
    例外 2 時の処理
```

ここでは、一般的書式を示すだけにしておきます (ソースコード 6.1)。

try のブロック中に、エラーが発生する可能性のある処理を記述します。except ブロックで、エラーが発生した場合の処理を書きます。エラーが発生しなければ、except の部分を実行することはありません。

エラーの種類に応じて対応策を切り分けることも可能です (ソースコード 6.2)。

7　課題

`quiz2.ipynb` にある課題を実施しなさい。

1 から 10 までの二乗の和を `for` を使って計算するプログラムを作成し、以下に記入しなさい。また、1 から n までの二乗の和は $n(n+1)(2n+1)/6$ となることと比較しなさい。

8　次回

プログラムを書く上で、データの塊を作って一括処理することはとても重要です。次回は、最も基本となるデータの塊であるリストを扱います。教科書では、6 章「リスト」です。