3. 条件分岐と繰り返し 1

Conditional Branching and Repetition 1

プログラミング・データサイエンス I

2024/4/25

1　今日の目的

今日の目的

- ここまでの例は、開始から終了まで一直線の処理
- 実際のプログラムでは
  - 条件にあうところだけ実行
  - 操作を繰り返す
  - 異常なことが起こった時の対処

ここまでのプログラム例では、プログラムの最初から最後へ向かって、基本的に、一直線に処理が進んでいきました。実際のプログラムでは、条件にあう部分だけを実行する、ある処理を繰り返す (これは、すこし出てきましたね)、異常が起こったら対応する、などが必要になります。現実世界の手続きでも、場合に応じた対応、同じことの繰り返し、さらに例外が発生した場合の処理があるのと同様です。

さて、今日のテーマに関連するサンプルプログラムを `git` を使ってダウンロードしてください。ダウンロードの仕方は、2 回目の講義で説明しました。忘れてしまった人は、参照してください。

`https://github.com/first-programming-saga/Control`

2 インデントとプログラムブロック: Indents and Program Blocks

インデントとプログラムブロック: Indents and Program Blocks

- Python では、インデントを使って、プログラムブロックを区別
  - 条件を満たした場合に実行する部分
  - 繰り返し部分
- プログラムブロック: プログラムの塊

条件分岐や繰り返しを記述するためには、条件を満たした場合に実行する部分、繰り返し実行する部分等を指定する方法が必要になります。それらのプログラムの部分が一行が書ければよいのですが、そうとは限りません。このような、プログラムの一部分のことを、プログラムブロック (program blocks) と言います。python は他のプログラミング言語とは非常に異なった方法で、プログラムブロックを指定します。ここが、初心者には分かりにくいところです。

インデント (indent) とは、行の先頭にある空白のことです。日本語でも英語でも、段落の先頭は一文字開けますね。これもインデントです。インデントすることで、視覚の上で、区切りやまとまりを示すことができます。python では、インデントすること及びその幅に特別な意味を与えます。そのため、不用意に空白を行の先頭に入れてはいけないのです。python では、インデントを使って条件を満たした場合に実行する部分、繰り返し実行する部分などのプログラムブロックを指定します。

3 条件分岐: Conditional Branching

3.1 条件分岐: Conditional Branching

条件分岐: Conditional Branching

- 条件に応じて、処理内容を分ける
  - 条件を満たす場合: `True`
  - 条件を満たさない場合: `False`

最初に、条件に応じて処理内容を分ける「条件分岐」について説明します。条件とは、比較 (ある変数がある値と等しい、あるいはより大きいなど) などを行った結果です。比

較などの結果は `True` か `False` という boole 型の値であることは、前回に説明しました。

3.2 条件分岐: 1

条件分岐: 1

- 条件を満たす場合の処理を指定
- インデントの使い方に注意

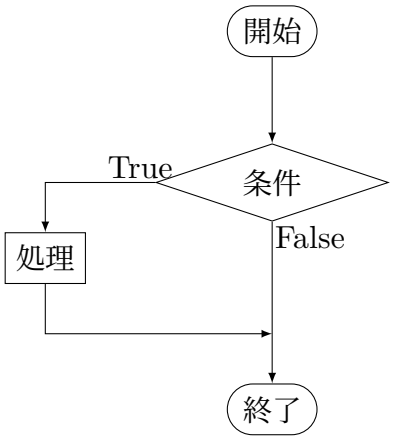

図 1　条件分岐の例 1。条件を満たす場合の処理を指定

非常に簡単な例を示します。図 1 は流れ図 (flow chart) という、処理の流れを図示したものです。ひし形の部分に条件を書き、長方形で処理を表します。図 1 は、条件を満たす場合には、なにかの処理を行い、満たさない場合には、何もしないという処理の流れを表しています。

ソースコード 3.1　簡単な条件分岐

```
1 a = - 10
2 if a < 0 :#a が負ならば、符号を変える
3     a *= -1
4 print(a)
```

`if.ipynb` を開いてください。最初のセルの部分を見てください (ソースコード 3.1)。2 行目で `a` という変数が負であるかを調べています。この条件を満たしたときは、3 行目を実行します。3 行目の先頭にスペースが 4 つあります。これがインデントです。なお、ス

ペースの数は 4 である必要はありません。ただし、同じプログラムブロックに属する部分は、同じインデント幅である必要があります。

もう一つ注意してください。2 行目の条件の後ろにコロン (:) があります (# から後ろはコメント)。これが新たなプログラムブロックの開始を表しています。なお、VSCode では、コロン (:) を入力して改行すると、自動的にインデントしてくれます。自分でインデントする場合には、通常は、タブキーを使います。

このプログラムの動作は予想できますか? a という変数が正ならば何もせず、負ならば符号を変え、いずれも新しい値を印刷します。`a *= -1` は `a = -1 * a` と同じ意味です。つまり、必ず正の値を印刷します。

3.3 条件分岐: 2

条件分岐: 2

- 条件を満たさない場合の処理も指定
- else の使い方

次は、if.ipynb の二番目のセルです (ソースコード 3.2)。条件を満たした場合と満たさない場合のそれぞれに処理がある場合です。対応するフローチャートを図 2 に示します。4 行目の `else:` で、「条件を満たさない場合」を指定しています。

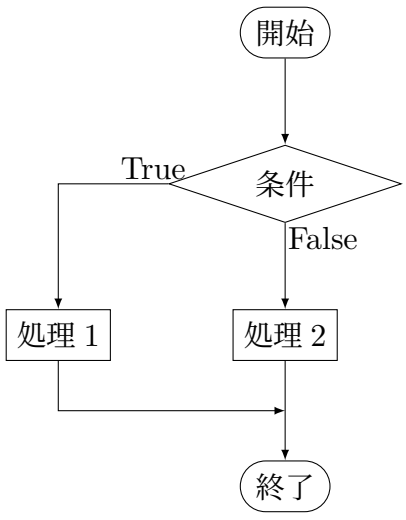

図 2　条件分岐の例 2。条件を満たさない場合の処理も指定

このプログラムの結果は、前のシートのものと同じです。違いは、ソースコード 3.2 では変数 a の値を直接印刷しているのではなく、別の変数 b に代入した後に印刷している

ソースコード 3.2　簡単な条件分岐 2

```
a = - 20
if a < 0 :#a が負の場合
    b = - a
else :#a が負でない場合
    b = a
print(b)
```

点です。

ソースコード 3.3　簡単な条件分岐 2

```
    a = - 20
    b = -1 if a < 0 else a
    print(b)
```

ソースコード 3.3 は、ちょっとスマートに変更したものです。動作は理解できますか。

3.4　条件分岐: 3

条件分岐: 3

- 条件に論理演算がある場合
- and、or、not
- 適切に () を使う

ソースコード 3.4　論理演算を含む条件分岐 1

```
message = '未設定'
if a > 0 and b > 0 :
    message = 'a も b も正です'
if a > 0 or b > 0 :
    message = 'a か b の少なくとも一方が正です'
print(message)
```

if.ipynb の 3 番目から 5 番目のセル (ソースコード 3.4 と 3.5) では、条件文に and や or が入っています。このように複数の条件を論理演算で結ぶことも可能です。

ソースコード 3.5　論理演算を含む条件分岐 2

```
message = '未設定'
if (a > 0 and b > 0) and c > 0 :
    message = 'a も b も c も正です'
if (a > 0 or b > 0) and c > 0 :
    message = 'a か b の少なくとも一方が正で、c は正です'
print(message)
```

5 番目のセル (ソースコード 3.5) では、三つの条件が書かれています。どのような順序で判定するのでしょう？ 判定の順序を明示するには、() を使います。() を書かないと、結合の強いものから順に評価をします。論理演算では、not が最も結合が強く、次が and、最も弱いのが or です。これを覚えておくのは大変ですから、() を使って、はっきりと書きましょう。条件が二つの場合でも () を使っても構いません。

3.5　条件分岐: 4

条件分岐: 4

- 条件文の入れ子構造
- elif の使い方

条件の中に、さらに条件がある場合があります。if.ipynb の 6 番目と 7 番目のセルを見てください。6 番目のセルは省略しています。ソースコード 3.6 です。図 3 も参照してください。

ソースコード 3.6　elseif の例

```
if a == b : #a と b が等しい場合
    c = a
    message = f'a=b={a}'
else: #a と b が異なる場合
    if a > b:
        c = a
        message = f'a={a} b={b} で a が大きい'
    else:
        c = b
        message =f'a={a} b={b} で b が大きい'
print(c, message)
```

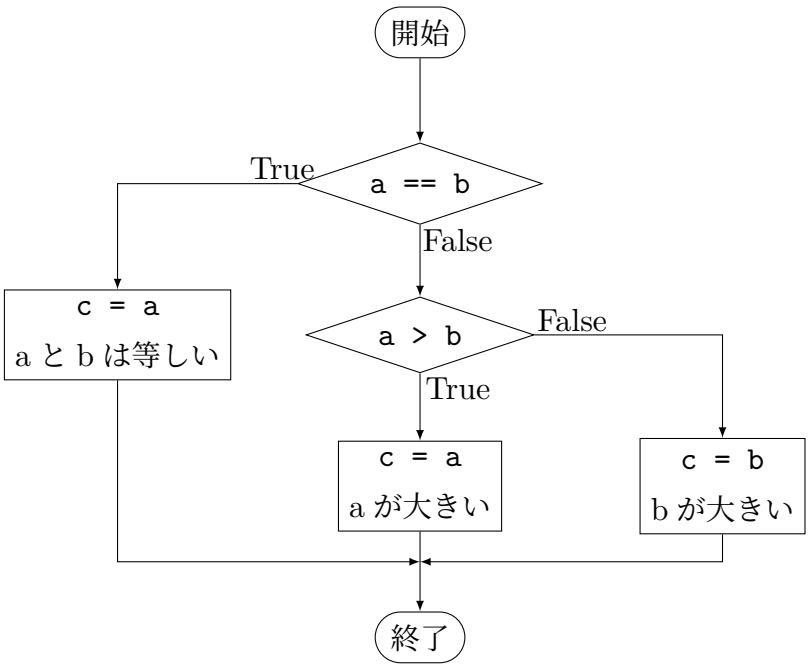

図 3　条件分岐の例 4。`elseif` の利用

`a` と `b` が異なる場合、つまり 4 行目の `else` で始まるブロック中で、更に `a` と `b` の大小関係を調べています。6 行目と 7 行目、9 行目と 10 行目のインデントがスペース 8 個になっています。これは、4 行目の `else` のさらに内側の 5 行目の `if` 、8 行目の `else` の部分に対するプログラムブロックであることを示しています。このように、インデントのが増えると、プログラムブロックの内部にプログラムブロックが入っていることを示します。

同じことを別の形式で表したものが、ソースコード 3.7 です。4 行目の `elif` は *else if* の意味で、1 行目の条件を満たさなかった場合に、更に条件を判定することを表しています。`elif` を使うと、プログラムブロックの入れ子にならないという利点があります。

課題 3.1　$x < 0$ の場合は $y = -1$、$0 \leq x < 1$ の場合は $y = 0$、それ以外は $y = 1$ となるプログラムを作成しなさい。また、その動作を確認しなさい。

4　`match` を使った場合分け

他の多くのプログラミング言語では、場合分けをする `switch` のような構文があります。しかし、python 3.10 以前では、場合分けをするには、`elif` を繰り返すしか方法が

ソースコード 3.7　elseif の例

```
if a == b : #a と b が等しい場合
    c = a
    message = 'a=b={a}'
elif a > b:
    c = a
    message = f'a={a} b={b} で a が大きい'
else:
    c = b
    message =f'a={a} b={b} で b が大きい'
print(c, message)
```

ありませんでした。elif の数が多くなると非常に読みにくくなってしまいます。python 3.10 で導入された場合分けの構文 match を見ていきましょう。

ソースコード 4.1　match の単純な例

```
x = 1
match x:
    case 0 | 1: # x=0 または x=1
        print(x)
    case 2:
        print(x)
    case _:# 値が一致しない場合
        print('default case')
        print(x)
```

ソースファイル match.ipynb を開けてください。最も簡単な例をソースコード 4.1 に示します。x の値によって、3 種類の処理をしています。各値の場合を示すのが case の部分です。3 行目では、x の値が 0 または 1 という二つの場合に適合する場合を表しています。|は、or を表しています。最後の case _は、どの場合にも一致しない場合です。

少し複雑な match の例をソースコード 4.2 と 4.3 に示します。ソースコード 4.2 では、後述する tuple という型の p という変数に対して、部分一致を使って場合分けをしています。tuple は、複数のオブジェクトを組にして保存することができます。ここで、定義している p は、2 次元空間の座標、つまり二つの float 型の値の組を表しています。それが、原点、x 軸、y 軸、それ以外を場合分けしています。

また、ソースコード 4.3 では、if を使って、一致する値の範囲を指定して、場合分けをしています。

ソースコード 4.2　match を使った部分一致の例

```
p = (1,0)
match p:
    case (0,0):
        print('origin')
    case (x,0):
        print(f'on x-axis: x={p[0]}')
    case (0,y):
        print(f'on y-axis: y={p[1]}')
    case _:
        print(p)
```

ソースコード 4.3　match を使った少し複雑な例

```
x=-2
match x:
    case 0:
        print(f'x は{x}')
    case x if x > 9:
        print('x は 10 以上')
    case x if x < 0:
        print('x は負')
    case _:
        print('x は一桁の自然数')
```

場合分けの数が少ないときには、if と else を使うのが標準的構文ですが、場合の数が多い場合には、match を使うことで見やすくなります。

5　while 文: while loops

5.1　条件を満たす限り繰り返す: Indefinite iterations

条件を満たす限り繰り返す: Indefinite iterations

- while の使用
- 繰り返し回数に注意

今日は、条件を使う構文をもう一つ説明します。「条件を満たす限り繰り返す」という構文です。英語のテキストで while 文の説明を読むと、回数を限定しない、つまり *indefinite*

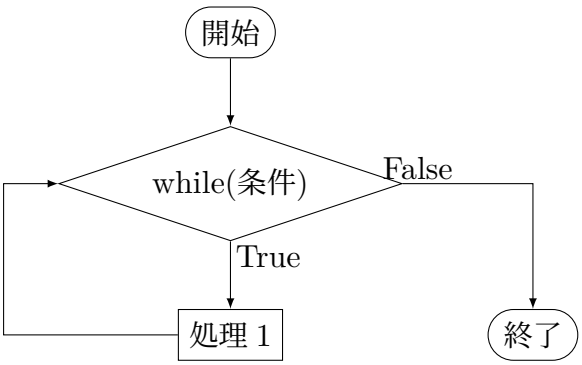

図 4　条件を満たす限り繰り返す

(無限定) な繰り返しという意味になっています。この繰り返しのことを、`while` ループとも言います。

図 4 を見てください。条件を満たす (`True`) と「処理 1」を行い、その後に再び条件判断の部分に戻ります。条件を満たさなくなる (`False`) と、終了します。つまり繰り返し回数に定めがなく、条件を満たす限り繰り返します。処理の中で、条件を満たすように何かが変化しているはずです。

`while0.ipynb` を開き、最初のセルを見てください (ソースコード 5.1)。初めに $c = 0$ とします。while ループの中で c を印刷するとともに、1 だけ増やします (4 行目)。やがて $c = 10$ となると条件を満たさなくなり、`while` ループを終了します。

ソースコード 5.1　簡単な `while` 文

```
1 c = 0
2 while c < 10:
3     print(c)
4     c += 1
```

二番目のセルの例をソースコード 5.2 に示します。

一つ注意しておきたいことがあります。`while` は条件を満たしている限り繰り返します。つまり、いつまでも終了しない「暴走」状態になる危険性があります。そのため、`while` を使う際には、プログラムを実行する前に、そのプログラムが停止することを確認しましょう。

それでも暴走してしまった場合には、メニュー中にある「...」を開き、その中の「Interrupt」または「割込み」を押して、強制的に停止します。

ソースコード 5.2　簡単な `while` 文

```
s = 0
c = 0
while s < 100:
    s += c
    print(f'c = {c}, s = {s}')
    c += 1
print(s)
```

5.2　`while` を使った例: Examples of `while` loops

Euclid 互除法: Euclidean Algorithm

- 二つの自然数の最大公約数を求める
- `while` の例

while の例として、二つの自然数の最大公約数を求める、Euclid の互除法を示します。高校の 1 年で習いましたか？

二つの自然数 m と n を与えます。n のほうが大きいとします。$m > 0$ である限り、以下を繰り返します。

1. n を m で割った余りを r とする
2. n に m を代入する
3. m に r を代入する

最後に得られた n の値が最大公約数となります。プログラムは、ソースコード 5.3 です。

5.3　break と continue

break と continue

- `while` ループの途中から抜ける方法
- break の使い方
- continue の使い方

繰り返し処理を行うプログラムでは、途中で繰り返しを止める、あるいはプログラムブロックの残りを実行せずに繰り返しの次の番に移りたいことがあります。それを実現する

ソースコード 5.3　Euclid 互除法

```
n = 465
m = 360
message = f'{n}と{m}の最大公約数は'
if m > n :#m>n の場合は入れ替え得る
    x = n
    n = m
    m = x

while m > 0 :
    r = n % m
    print(f'{n} % {m} = {r}')
    n = m
    m = r

print(f'{message}{n}')
```

方法の一つが、break と continue を使うことです。

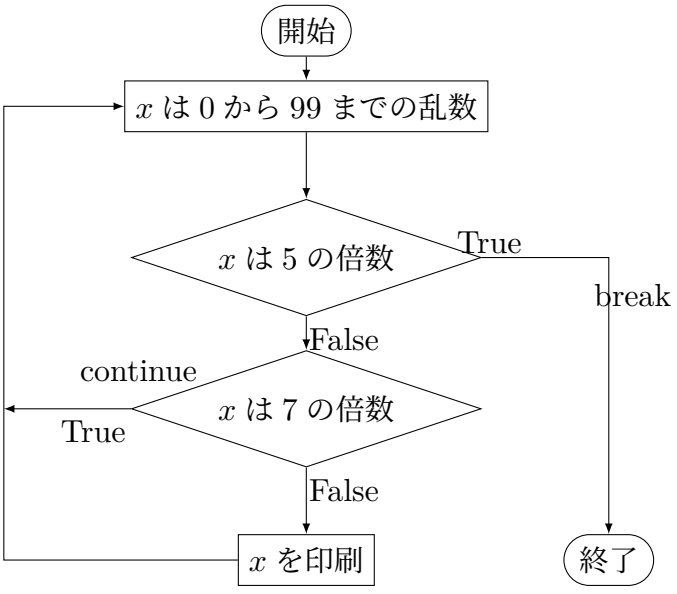

図 5　乱数を生成し、場合分けをする例

while.ipynb を開けてください。break と continue について説明します (ソースコード 5.4)。処理の流れに注意してください。大まかにいうと、break は繰り返し処理を中断することを、continue は繰り返し処理を継続することを表します。

これらの方法を使わなくても、他の方法で同様のことはできます。そのため、「こうい

うこともできる」という程度の理解で十分です。

例を見てください。while ループは条件に True とありますから、無限ループです。その次の行で、x には、0 から 99 までのでたらめな整数が入ります。x が 5 で割り切れたら while ループを抜けるというのが break です。x が 7 で割り切れたら、それ以降の while 内の処理を行わず、while の先頭に戻るというのが continue です。流れ図を図 5 に示します。

ソースコード 5.4　乱数を生成し、場合分けをする例

```
1 while True:
2     x = randint(0,99)
3     if x % 5==0:
4         break#while ループを抜ける
5     elif x % 7 == 0:
6         continue#while ループの先頭へ
7     print (x)
8 print("end with "+str(x))
```

5.4　while と else

while と else

- while ループが中断しなかった場合の処理
- 素数か否かの判断の例

break は while ループを中断して外へ出てしまいます。中断せずに while ループの最後まで至った時の処理を記載するのが、while の後の else です。図 6 の条件 1 が、while の条件です。while 内の条件 2 で break が発生せず、while の条件が満たされない場合に、else で指定された処理を実行します。

具体例を見ましょう。ソースコード 5.5 は、自然数 n が素数かを判定するプログラムです。まず、奇数以外の場合を match を使って場合分けしています。

11 行目以降では、n は偶数ではないことが分かっています。ある奇数が素数かを判定するのは、非常に手間がかかります。その方法は、3 から始めて $\sqrt{n}$ まで順に奇数で割ってみるしかありません。途中で割り切れたら、素数ではないことが分かります。割り切れなかったら素数となります。割り切れる場合には、while ループの途中で終わります。割り切れないと、while ループが完了し、else の部分に至って、素数であるというメッセー

ソースコード 5.5　素数か否かの判定

```
n = 219
match n:
    case n if n <= 0:
        print(f'{n}は正でなければならない')
    case n if n < 2:
        print(f'{n}は素数ではない')
    case 2:
        print(f'{n}は素数である')
    case n if n % 2 == 0:
        print(f'{n}は偶数であり、素数ではない')
    case _:
        m = int(math.sqrt(n))
        k = 3
        while k <= m:
            if n % k == 0:
                print(f'{n}は{k}で割り切れるため、素数ではない')
                break#while ループから抜け出す
            k += 2
        else:#while ループの最後まで至った場合
            print(f'{n}は素数である')
```

ジを出します。手順をフローチャートでも示しています (図 7)。

最後に、もう一度、while.ipynb の先頭を見てください (ソースコード 5.6)。前回の最後に、関数とライブラリの話しを少しだけ紹介しました。while.ipynb の最初の部分で、二つの Python 標準ライブラリをインポートしています。

最初の部分は、random というライブラリから randint という関数を導入している部分です。この関数で、でたらめな整数を生成しています。

二番目は、math という数学関数のライブラリを導入しています。平方根を計算する関数 math.sqrt() を使うためです。

ソースコード 5.6　ライブラリのインポート

```
from random import randint #乱数生成に必要なモジュール
import math #数学関数のモジュール
```

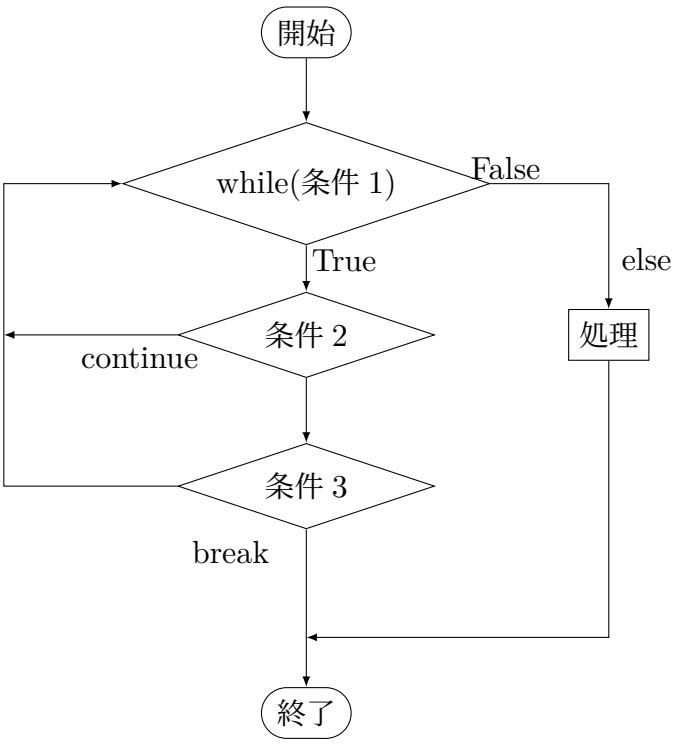

図 6　`while` と `else`

6　課題

`quiz1.ipynb` にある課題です。

1 から 10 までの和を `while` を使って計算するプログラムを `while.ipynb` の末尾に作成し、以下に記入しなさい。また、1 から n までの和は $n(n+1)/2$ となることと比較しなさい。なお、無限ループにならないように十分に注意しなさい。

7　次回

次回は、5 章の後半、`for` を使った繰り返しと例外処理を扱います。

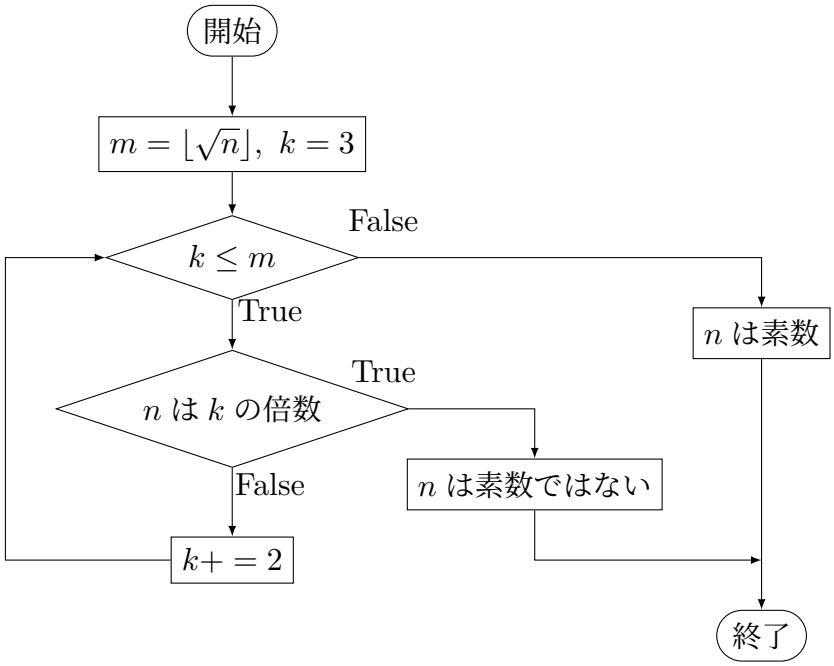
開始
$m = \lfloor \sqrt{n} \rfloor,\ k = 3$
$k \leq m$
False
True
n は素数
n は k の倍数
True
False
n は素数ではない
$k+ = 2$
終了

図 7　素数か否かの判定