

Python プログラミング基礎 教材 3

九州大学 数理・データサイエンス教育研究センター

全体の構成

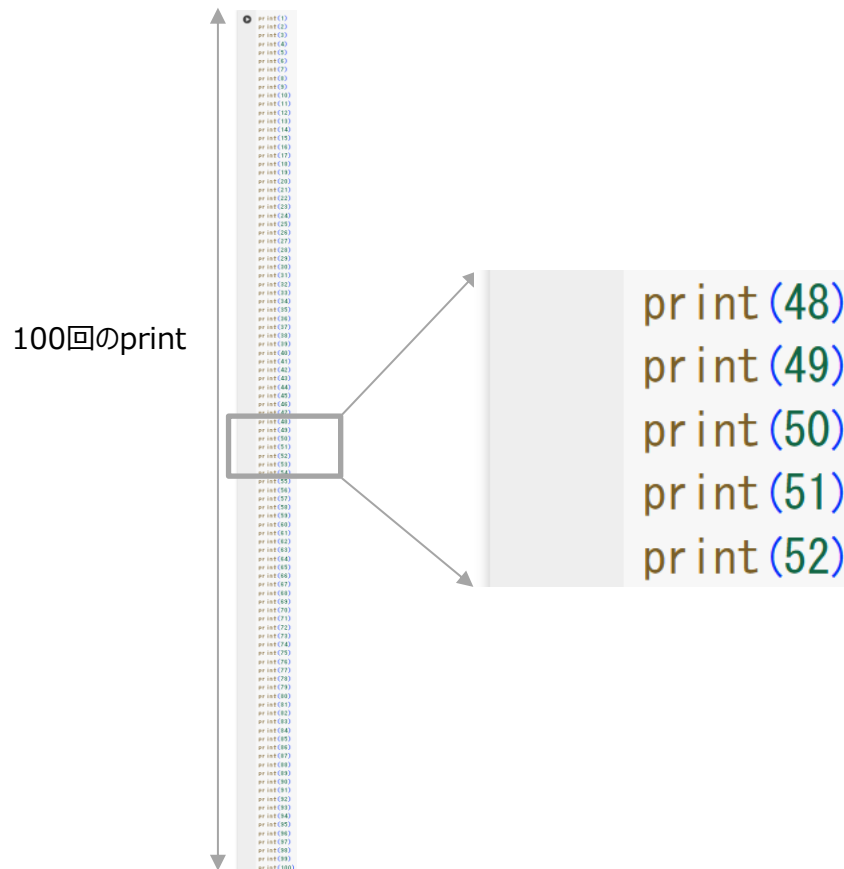
- 教材 1
 - インTRODクション～プログラムってなんだ？
 - プログラミング言語 Python ～無料で遊べるのに凄い
 - 【準備】Gogole Colaboratoryへ行こう！
 - 文字を表示してみよう ～ 記念すべき最初のプログラム
- 教材 2
 - 計算させてみよう ～ たす, ひく, かける, わる
 - 変数 ～ 電卓を越える！
 - If文 ～ ますますプログラムっぽく
 - 便利なコメントアウト ～ ちょっと休憩
 - 数当てゲームを作ってみよう ～ ゲームの基本中の基本
- 教材 3
 - For文 ～ 人間には面倒な繰り返しを簡単に
 - 【寄り道】コンピュータの計算精度 ～ 意外と正確じゃない
 - 再びゲームを作ってみよう ～ 工夫次第でどんどん楽しく
- 教材 4
 - List ～複数の変数を固めたもの
 - 関数とライブラリ ～ ひみつ道具で遊ぼう
- 付録
 - if文のちょっと進んだ使い方
 - AIを用いたプログラミング
 - Google Driveとの連携

for文

人間には面倒な繰り返しを簡単に

ひどいミッション： 「数字を1から100まで」を表示せよ

- これまでのやり方なら…



面倒くさすぎる…
抜けがないかも心配だし、
1000や10000までとなると、
もう無理…



発想の転換～変数を使えば手抜きができる!?

1,2,3,...,100それぞれを
print(0), print(1),... print(100)で表示



変数*i*の中身を1から100まで変えながら
print(*i*)を繰り返せないか？



for文を使ってシンプルに表現

- 100回もprintを書かなくても、たった2行でOK!



```
for i in range(100):  
    print(i)
```

ここでも登場
インデント

“for”が出てくるところを詳しく



```
for i in range(100):
    print(i)
```



変数 *i*

1から100までの数の集まり

```
for i in range(100):
```

~に対して

~の中の

for文の最後には
“:” (コロン)を付ける
(pythonのルール)




1から100までの数を
順次代入した変数*i*に対して
インデント部分を繰り返す

※実はちょっと
ウソがあります。
その辺は数枚後に

ということは…!?

iを1から順に100まで変化させながら
インデント部分(変数iの中身を表示)を繰り返す



```
for i in range(100):  
    print(i)
```

※ここもちょっと
ウソがあります。
その辺は数枚後に

これなら1000まででも10000まででも
表示は楽勝！



ここを1000や10000に
するだけで完了



```
for i in range(100):  
    print(i)
```

あれ？ 変だぞ… 実行結果をよく見ると…



```
for i in range(100):  
    print(i)
```



0
1
2

:

96
97
98
99

1からじゃなくて0から始まっている

100じゃなくて99で終わっている

確かに100回
繰り返しては
いるが…

実は `range(100)` は、1から100までではなく 0から99まで！

- これはpythonの（ちょっと不思議な）ルールです
 - そういふもんだと覚えるしかない！
- 正確に1から100まで表示させようとするならば、



```
for i in range(100):  
    print(i+1)
```

変数*i*の中身に1をプラスして表示



1
2

1から始まっている

⋮

99
100

100で終わっている

range(100)以外の繰り返しの指定法

● 範囲指定タイプ

- `range(100)` → 0, 1, 2, ..., 98, 99
- `range(50, 100)` → 50, 51, ..., 98, 99 #開始番号指定
- `range(0, 100, 5)` → 0, 5, 10, 15, ..., 90, 95 #開始と刻み幅
- `range(10, 0, -1)` → 10, 9, 8, ..., 2, 1 #負の刻み幅による逆順

● リストタイプ ([]内に書かれたものが書かれた順番通りに)

- `[1, 2, 3, 5, 7, 11]` → 1, 2, 3, 5, 7, 11
 - `[5, 1, 2, -10, 6]` → 5, 1, 2, -10, 6
 - `["みかん", "りんご", "バナナ"]` → "みかん", "りんご", "バナナ"
 - `["みかん", 5, "りんご", 3, "バナナ", 1]` → "みかん", 5, "りんご", 3, "バナナ", 1
- なので, `range(5)`と`[0,1,2,3,4]`は同じ
- 前者のほうが書くのは楽ですね

練習：（あまり面白くないですが） 九九の「7の段」を表示するプログラムを作ってみよう



```
for i in range(1, 10):  
    print(i*7)
```



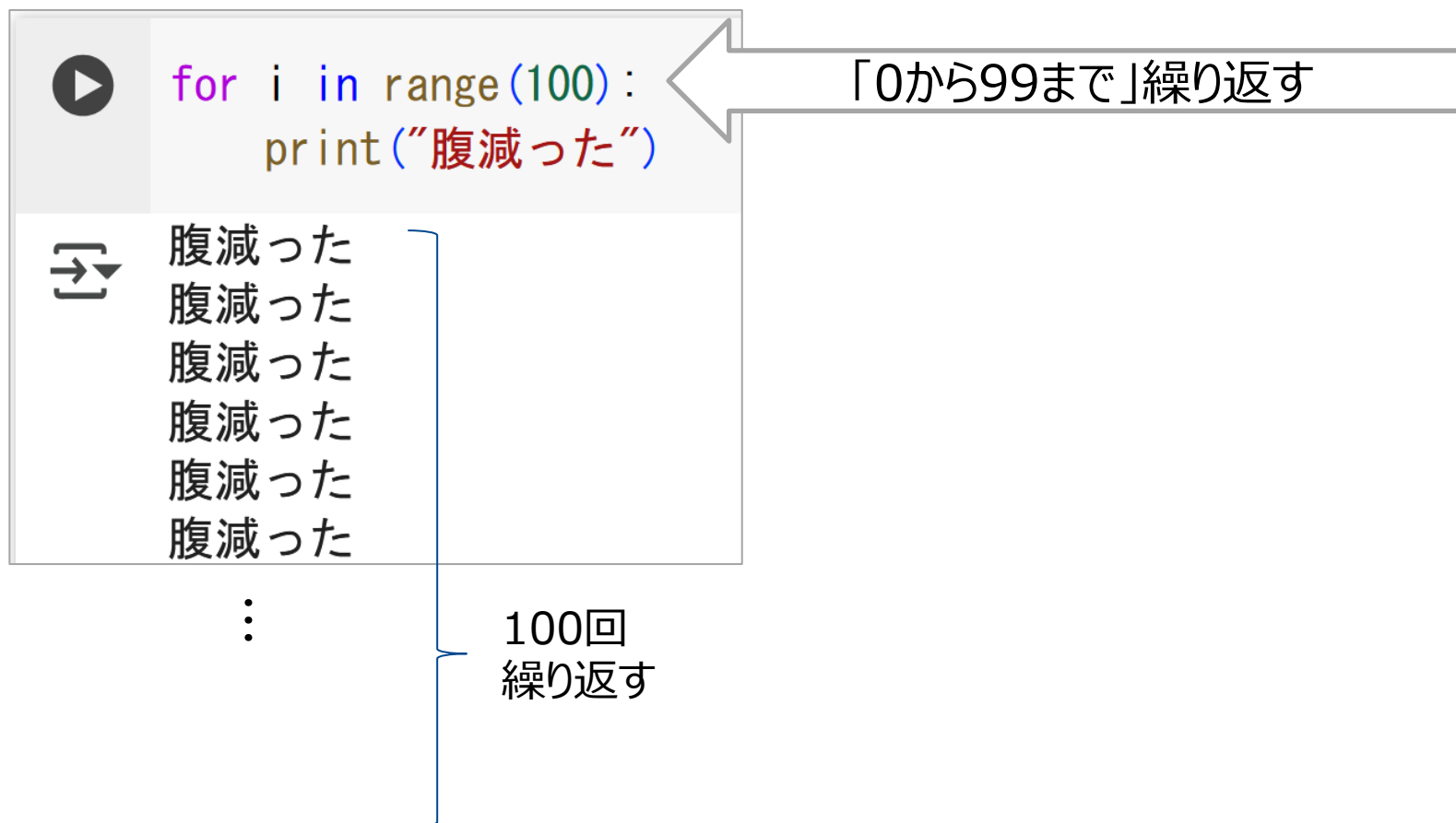
```
7  
14  
21  
28  
35  
42  
49  
56  
63
```

「1から9まで」を繰り返すように指定

- 簡単ですね！
 - iは最大9までなので、70は出てこない

単なる繰り返しにも利用できる

- 以下の例, どちらもインデント部分でiの値は参照していないが, それはそれでOK



The image shows a code editor window with a play button icon on the left. The code is:

```
for i in range(100):  
    print("腹減った")
```

Below the code, the output is shown as a list of six lines, each containing the text "腹減った". A vertical ellipsis follows, and a bracket on the right indicates that the loop repeats 100 times.

A callout box with a white background and a grey border points to the `range(100)` part of the code. It contains the text "「0から99まで」繰り返す".

繰り返されるのは インデントされた部分のみ

▶

```
for i in range(100):
    print("腹減った")
    print("ごはんまだ?")
```

⇒

```

腹減った
ごはんまだ?
腹減った
ごはんまだ?
腹減った
ごはんまだ?
    ⋮
腹減った
ごはんまだ?
  
```

2行分を100回
(すなわち計200行分)
繰り返す

▶

```
for i in range(100):
    print("腹減った")
print("ごはんまだ?")
```

⇒

```

腹減った
腹減った
腹減った
腹減った
    ⋮
腹減った
腹減った
腹減った
ごはんまだ?
  
```

1行分を100回
(すなわち計100行分)
繰り返す

for文の中にfor文を： 九九のプログラム (1/2)



```
for i in range(1, 10):
```

```
    for j in range(1, 10):
```

```
        print(i*j)
```

iについて「1から9まで」を繰り返すように指定

「iの繰り返しの中」で「jも1から9までを繰り返す」ように指定



```
1
2
3
4
5
6
7
8
9
2
4
6
8
10
⋮
```

1の段 (iが1のまま, jが1から9まで)

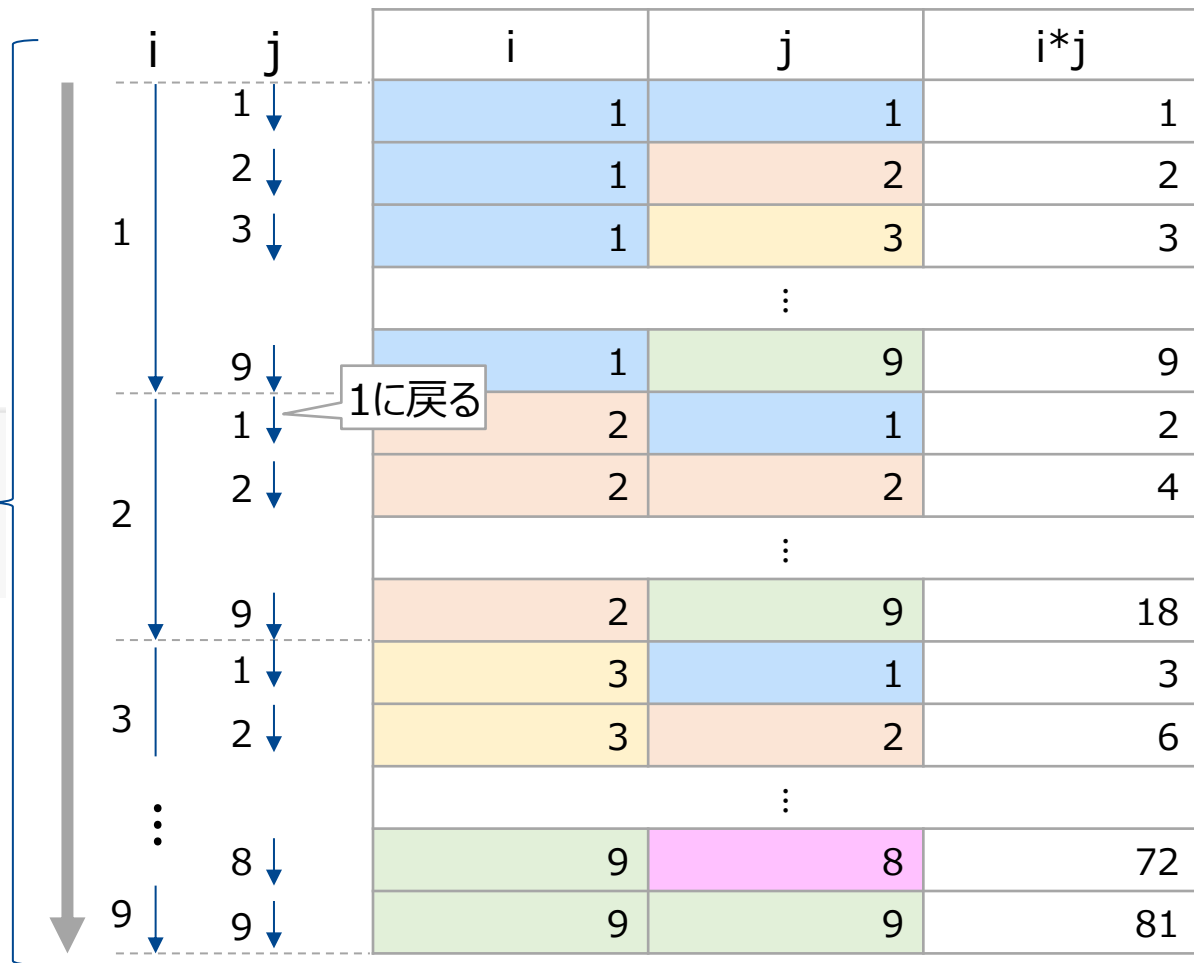
2の段 (iが2のまま, jが1から9まで)

for文の中にfor文を： 九九のプログラム (2/2)

- 繰り返しの順で変数*i*,*j*の中身はどう変わっていく？



```
for i in range(1, 10):
    for j in range(1, 10):
        print(i*j)
```



練習：（あまり面白くないですが） 九九を表示するプログラムを自分でも作ってみよう

- といっても、この通り作ればOK



```
for i in range(1, 10):
    for j in range(1, 10):
        print(i*j)
```

• 応用編

- 九九じゃなくて「十十」はどうする？
 - ヒント：range(1,10)をrange(1,11)とする
- 特定の段だけスキップする
 - ヒント：range(1,10)の代わりに、[1,2,4,5,6,7,8,9]を使う
- 逆九九を作ってみる
 - ヒント range(1,10)をrange(9,0,-1)に
- 特定の段(例：3の段)だけ，“内緒”と表示させる
 - ヒント：if i==3: else: を使って，表示法を使い分ける
- 九九じゃなくて「九九九」はどうする？
 - ヒント：forのなかのforのなかにまたforが

もうちょっと凝ったfor文： 1から100まで足す計算（1/3）

- 完成版を先に見てみましょう



```
s = 0
for i in range(1, 101):
    s = s + i
print(s)
```



5050

もうちょっと凝ったfor文： 1から100まで足す計算（2/3）

● 解説



```
s = 0
```

```
for i in range(1, 101):
```

```
    s = s + i
```

```
print(s)
```

変数sを準備し、中身をゼロに「初期化」

変数iを1から100まで

現在のsの値にiを加えて、またsに代入

なぜこれが必要かは後述



5050

このスライド
覚えてますか？

補足：「何かへんだな？」と思った人は鋭い

- 先ほどの式、よく見るとなんか変だ…

```
[3] MonsterHP = MonsterHP - 40
```

- 数学で習ってきた「等式」なら、右辺と左辺は絶対「イコール (=)」にはならないはず…

・ある数から40引いたものが、元の数に等しいなんてことはあり得ない

- 実はこのプログラムにおいて、「=」はイコールではなく、代入を意味する

```
[3] MonsterHP = MonsterHP - 40
```

①元のMonsterHPの中身を呼び出し

②それから40をマイナスする

③その結果を左辺(MonsterHP)に代入する

- 右の①②③の手順と解釈するのが正解

・まず右辺を処理し、その結果を左辺に代入

もうちょっと凝ったfor文： 1から100まで足す計算（3/3）

- 実際にどのように計算が進むのか？

```

▶ s = 0
  for i in range(1, 101):
    s = s + i
  print(s)

```

⇒ 5050

変数 i	右辺の s (現在の s)	左辺の s (新しい s)
1	0	1
2	1	3
3	3	6
4	6	10
5	10	15
6	15	21
100	4950	5050

初期化がなぜ必要なのか？ (1/2)



```
s = 0
```

```
for i in range(1, 101):
```

```
    s = s + i
```

```
print(s)
```



```
5050
```

これが初期化

変数 i	右辺の s (現在の s)	左辺の s (新しい s)
1	0	1
2	1	3
3	3	6
4	6	10
5	10	15
6	15	21
100	4950	5050

この部分は
この最初の値
(初期値)
があるから
順次計算
できる



初期化がなぜ必要なのか？ (2/2)

- 初期化せずに動かそうとすると叱られる…

```

▶ for i in range(1, 101):
    s = s + i
    print(s)

```


```

NameError
<ipython-input-2-e5842d50b36b> in <cell>:1
----> 1 for i in range(1, 101):
      2     s = s + i
      3     print(s)

NameError: name 's' is not defined

```

さって何？ それがわからないから (最初の) s+が計算できないよ！



次のステップ: [エラーの説明](#)

最初が決まらないと
あとが計算できない

変数 i	現在の s (現在の s)	左辺の s (新しい s)
1	?	??
2	???	????
3		
4		
5		
6		
100		

- だから、明示的に「sは0からスタート」させるという初期化は大事
 - なお i については、range(1,100)により最初に1が入ることが明示されている→別途の初期化不要

ところで…叱られるだけまし！

- 今回はエラーが出て、動きませんでした（なんの結果も出なかった）

```

▶ for i in range(1,101):
  s = s + i
  print(s)

```

```

NameError                                Traceback (most recent call last)
<ipython-input-2-e5842d50b36b> in <cell line: 0>()
      1 for i in range(1,101):
----> 2     s = s + i
      3     print(s)

NameError: name 's' is not defined

```

次のステップ: [エラーの説明](#)

さって何？それがわからないから
(最初の) $s+i$ が計算できないよ！

「叱られる
うちが華」だな。
気を付けよう…



叱られなかったから、
大嘘の答えが出ているのに
気づかなかった!!

- しかし、エラーがでてくれた「おかげ」で初期化を忘れていることに気づけました
- 一方、（別の計算かなにかで）変数sにすでに何かの値が入っていたら（例えば100）、叱られることもなくウソの答（5150）が出てくる！これが最悪！
- 本当に怖いのは「一見正しく動いているように見えるプログラム」





練習：借金がどのように膨れるか？

- 借金の利子の付き方
 - 単利：元々借りた金額に，毎年 $a\%$ の利子がつく
 - 複利：その年の借りている金額に， $a\%$ の利子がつく
- 式で書くと…
 - 単利： $\text{amount} = \text{amount} + \text{元々借りた金額} \times a\%$
 - amountの初期値 = 元々借りた金額
 - 複利： $\text{amount} = \text{amount} + \text{amount} \times a\%$
 - amountの初期値 = 元々借りた金額
- 複利の場合，こんなプログラムに（是非，自分でも打ち込んでみよう）



```
org = 100000 # 元々10万円借りた
interest = 0.1 # 利子10%
amount = org # 支払額を元々借りた金額で初期化
for i in range(10):
    amount = amount + amount * interest
print(f"after {i+1}-year, amount becomes {amount}")
```

10年分計算してみる

借金額を毎年アップデート

借金額の表示例

プログラムをタイプするのが面倒な方へ：
前ページの「練習」コピペ用ページ



```
org = 100000 # 元々10万円借りた
interest = 0.1 # 利子10%
amount = org # 支払額を元々借りた金額で初期化
for i in range(10):
    amount = amount + amount * interest
    print(f"after{i+1}-year, amount becomes {amount}")
```

Google Colabにペーストできない場合、windowsなら一旦メモ帳にペーストして、そこから再度コピペするとうまくいくかも

もしかしたら、インデント部分にこんな謎の黄色い□が出てくるかも。それは消そう。

```

↓ ↓
□ □ answer = random.randint(1,
□ □ guess = int(input("1~60
```

練習：借金がどのように膨れるか？ 実行してみると…



```
▶ org = 100000 # 元々10万円借りた
  interest = 0.1 # 利息10%
  amount = org # 支払額を元々借りた金額で初期化
  for i in range(10):
    amount = amount + amount * interest
    print(f"after {i+1}-year, amount becomes {amount}")
```

```
⇒ after1-year, amount becomes 110000.0
   after2-year, amount becomes 121000.0
   after3-year, amount becomes 133100.0
   after4-year, amount becomes 146410.0
   after5-year, amount becomes 161051.0
   after6-year, amount becomes 177156.1
   after7-year, amount becomes 194871.71000000002
   after8-year, amount becomes 214358.88100000002
   after9-year, amount becomes 235794.76910000003
   after10-year, amount becomes 259374.24601000003
```

応用①：
利子や年数を変えて、
10万円の借金が
どう膨れ上がるか
見てみよう

100年後の金額を見ると、
絶対びっくりすると思いますよ！
年利10%恐るべし！

応用②：
単利の計算にしてみよう



10年で2.6倍に！（借金地獄！）

【寄り道】 コンピュータの計算精度

意外と正確じゃない

ところで…先ほどの結果, なんだか変だと思いませんか？

```
▶ org = 100000 # 元々10万円借りた
interest = 0.1 # 利子10%
amount = org # 支払額を元々借りた金額で初期化
for i in range(10):
    amount = amount + amount * interest
    print(f"after {i+1}-year, amount becomes {amount}")
```

```
⇒ after1-year, amount becomes 110000.0
after2-year, amount becomes 121000.0
after3-year, amount becomes 133100.0
after4-year, amount becomes 146410.0
after5-year, amount becomes 161051.0
after6-year, amount becomes 177156.1
after7-year, amount becomes 194871.71000000002
after8-year, amount becomes 214358.88100000002
after9-year, amount becomes 235794.76910000003
after10-year, amount becomes 259374.24601000003
```

あれ？
なんか桁数が
急に増えてる…

もっとびっくりするかも…



0.1+0.2



0.3ぴったり
じゃないの!?



0.300000000000000004

なので、こんなことも生じます

```

▶ a = 0.1+0.2
  if a == 0.3:
    print("等しい")
  else:
    print("等しくない")

```

⇄ 等しくない

等しいだろ!

参考記事「0.1+0.2≠0.3」を説明できないエンジニアがいるらしい
https://qiita.com/higashi_nc/items/9a5ea00415a008f06843

コンピュータで すべての数が正確に表せるわけではない！(1/2)

● 整数だけを扱うなら(異常に大きい数でない限り) 大丈夫

- 正確に計算できるので安心
 - なお、整数でも割り算が出てくると小数が出てくるので、以下の問題に注意

● 問題は**小数**が出てくるとき！

- 実はコンピュータは0.1ですら正確に表現できない
- 理由は、コンピュータがすべての数字を0と1の組み合わせ（二進法）で表すから
 - より詳細な理由は本資料の範囲外なので割愛
 - 参考：「 $0.1+0.2 \neq 0.3$ 」を説明できないエンジニアがいるらしい https://qiita.com/higashi_nc/items/9a5ea00415a008f06843
 - 「 $1/3$ が0.3333…と永遠に続くので正確な小数で表せない」ことと根は同じ

● コンピュータで**小数を扱うときは「誤差が含まれうる」ことに常に留意しよう**

- 誤差により…

変な端数が出て着たり…

```
after5-year, amount becomes 161051.0
after6-year, amount becomes 177156.1
after7-year, amount becomes 194871.71000000002
after8-year, amount becomes 214358.88100000002
after9-year, amount becomes 235794.76910000003
after10-year, amount becomes 259374.24601000003
```

イコールが成り立たなかったり…

```
▶ a = 0.1+0.2
  if a == 0.3:
    print("等しい")
  else:
    print("等しくない")
```

⇄ 等しくない

コンピュータで すべての数が正確に表せるわけではない！ (2/2)

- 現実的な対処法
 - 「まあそんなもんだ」とあきらめる
 - 小数点以下を適当なところで四捨五入して，表示だけは整える（=誤差をごまかしている）
 - より高度な方法(例えば decimalモジュール)を使って計算する
- 参考：その「より高度な方法」で先ほどの借金プログラムを書くと…

```

▶ from decimal import Decimal, getcontext

getcontext().prec = 20 # 計算精度（桁数）の設定

org = Decimal("100000") # 元々10万円借りた
interest = Decimal("0.1") # 利子10%
amount = org # 支払額を元々借りた金額で初期化

for i in range(10):
    amount = amount + amount * interest
    print(f"after {i+1}-year, amount becomes {amount}")
  
```

結果

```

⇒ after1-year, amount becomes 110000.0
   after2-year, amount becomes 121000.00
   after3-year, amount becomes 133100.000
   after4-year, amount becomes 146410.0000
   after5-year, amount becomes 161051.00000
   after6-year, amount becomes 177156.100000
   after7-year, amount becomes 194871.7100000
   after8-year, amount becomes 214358.88100000
   after9-year, amount becomes 235794.769100000
   after10-year, amount becomes 259374.2460100000
  
```

先ほどの結果

```

after7-year, amount becomes 194871.71000000002
after8-year, amount becomes 214358.88100000002
after9-year, amount becomes 235794.76910000003
after10-year, amount becomes 259374.24601000003
  
```

再びゲームを作ってみよう！

工夫次第でどんどん楽しく

ゲームを作る材料

- 変数
- print
- if
- else
- random
- input
- for

これまでに学んだツール





ゲーム 1 : 数当てゲームを5回繰り返す

```

▶ import random

for i in range(5):
    answer = random.randint(1, 6)
    guess = int(input("1~6の数字を当ててみて! : "))
    if guess == answer:
        print("正解!")
    else:
        print(f"はずれ! 本当は {answer}")
  
```

先ほどの
数当てゲームが
for文の中に入っただけ

実行例

```

⇒ 1~6の数字を当ててみて! : 2
   はずれ! 本当は 5
   1~6の数字を当ててみて! : 4
   はずれ! 本当は 1
   1~6の数字を当ててみて! : 1
   はずれ! 本当は 3
   1~6の数字を当ててみて! : 1
   はずれ! 本当は 4
   1~6の数字を当ててみて! : 4
   正解!
  
```

練習：数当てゲームを5回繰り返しつつ、何回当たったかを表示する



```
import random
```

```
atari = 0 #当たった回数. 0で「初期化」
for i in range(5):
    answer = random.randint(1, 6)
    guess = int(input("1~6の数字を当ててみて!: "))
```

追加. 初期化の必要性は,
「1から100まで足す計算」の話を参照

```
if guess == answer:
    print("正解!")
    atari = atari + 1 #当たった回数を増やす
else:
    print(f"はずれ! 本当は {answer}")
```

追加. 当たった場合のみ,
atariを1つ増やす.
右辺を左辺に代入.

```
print(f"5回中 {atari} 回当たりました")
```

追加. 結果表示

実行例

```
⇒ 1~6の数字を当ててみて!: 2
はずれ! 本当は 1
1~6の数字を当ててみて!: 2
正解!
1~6の数字を当ててみて!: 3
はずれ! 本当は 6
1~6の数字を当ててみて!: 1
はずれ! 本当は 6
1~6の数字を当ててみて!: 4
はずれ! 本当は 1
```



プログラムをタイプするのが面倒な方へ：
前ページの「練習」コピペ用ページ

```
import random
```

```
atari = 0 #当たった回数. 0で「初期化」
```

```
for i in range(5):
```

```
    answer = random.randint(1, 6)
```

```
    guess = int(input("1~6の数字を当ててみて!"))
```

```
    if guess == answer:
```

```
        print("正解!")
```

```
        atari = atari + 1 #当たった回数を増やす
```

```
    else:
```

```
        print(f"はずれ! 本当は {answer}")
```

```
print(f"5回中 {atari}回当たりました")
```



自主練習：いろいろ変えて遊んでみよう

- アイディア① サイコロの出る目を i に応じて変えてみよう
 - ヒント：`randint(1,6)`を`randint(1,i)`にしてみる？
- アイディア② はずれたら当たり回数をゼロリセットする鬼ルールに
 - ヒント：else時に `atari = 0`
- アイディア③ 数字1を1と当てられたら、当たり回数を+10
 - ヒント：`atari=atari+1`の後に、「if `answer==1` なら `atari = atari+10`」
- アイディア④ 一度でも当てられたら、1か2しか出ないイージーモードに
 - ヒント：`randint(1,k)`として、 k は6で初期化。 `atari=atari+1`の後に、`k=2`
- アイディア⑤ 2回連続で当たったら当たり回数を+10
 - ヒント：実はこれはちょっと難しい。わかるかな...？
 - 次のスライドに答えを...

プログラムをタイプするのが面倒な方へ：
 「2回連続で当たったら当たり回数を+10」
 コピペ用ページ



```
import random

atari = 0
success = 0 #直前が当たりなら1に. はずれなら0
for i in range(5):
    answer = random.randint(1, 6)
    guess = int(input("1~6の数字を当ててみて!: "))

    if guess == answer:
        print("正解!")
        atari = atari + 1
        if success == 1: #直前も当たっていた!
            atari = atari + 10 #ボーナスで+10
            success = 1 #当たったので1に
    else:
        print(f"はずれ! 本当は {answer}")
        success = 0 #はずれたので0に

print(f"5回中 {atari}回当たりました")
```

2回連続で当てるのって
 レアなので、プログラムが
 ちゃんと動いているのか
 確認するのが大変…

それならrandint(1,2)と
 して当てやすくしておく
 と、確認しやすくなるよ



for文を途中で抜ける: break

- 例：全5回繰り返さなくても，当てた時点でもう止めたい…

```

▶ import random

for i in range(5):
    answer = random.randint(1, 6)
    guess = int(input("1~6の数字を当ててみて!"))

    if guess == answer:
        print("正解")
        break
    else:
        print(f"はずれ! 本当は {answer}")

print(f"{i+1}回で終了!")

```

正解した時点でfor文を抜ける!

forを抜けた時点でのi (+1)の値を表示

⇒ 1~6の数字を当ててみて!:1
 はずれ! 本当は 6
 1~6の数字を当ててみて!:1
 正解!
 2回で終了!

range(5)としていても，途中で抜けた場合は変数も4 (=5-1) に到達せずに途中の値となる

ゲーム2: モンスターと3回勝負する ロールプレイング風ゲーム



```

▶ import random

HP = 10 # 勇者の HP
print(f"あなたは勇者です。HPは{HP}")
print("モンスターがあらわれた!")

for i in range(3): # 3回勝負
    print(f"残りHP {HP}")
    player = random.randint(1, 6)
    monster = random.randint(1, 6)

    print(f"あなたの攻撃 {player}")
    print(f"モンスターの攻撃 {monster}")

    if player > monster: # あなたの攻撃が強かった
        print(f"やった! HPが{player}増えた!")
        HP = HP + player
    else: # モンスターの攻撃が強かった
        print(f"やられた! HPが{monster}減ってしまった")
        HP = HP - monster

if HP < 0:
    print("3回勝負の結果, HPが残ってない... あなたの負けだ")
else:
    print(f"HPはまだ{HP}だ! 生き残ったあなたの勝ち!")

```

3回戦いを繰り返す!

自分の攻撃が強ければ、
HPがそれだけ増える

モンスターの攻撃が強ければ
HPがそれだけ減る

戦い終わったあと、HPが
残ってるかどうかでメッセージを
変える

練習：モンスターと3回勝負する ロールプレイング風ゲーム



```

▶ import random

HP = 10 # 勇者の HP
print(f"あなたは勇者です。HPは{HP}")
print("モンスターがあらわれた!")

for i in range(3): # 3回勝負
    print(f"残りHP {HP}")
    player = random.randint(1, 6)
    monster = random.randint(1, 6)

    print(f"あなたの攻撃 {player}")
    print(f"モンスターの攻撃 {monster}")

    if player > monster: # あなたの攻撃が強かった
        print(f"やった! HPが{player}増えた!")
        HP = HP + player
    else: # モンスターの攻撃が強かった
        print(f"やられた! HPが{monster}減ってしまった!")
        HP = HP - monster

if HP < 0:
    print("3回勝負の結果, HPが残ってない... あなたの負けだ")
else:
    print(f"HPはまだ{HP}だ! 生き残ったあなたの勝ち!")

```

自分で書いて
動かしてみよう!

実行例

```

⇒ あなたは勇者です。HPは10
   モンスターがあらわれた!
   残りHP : 10
   あなたの攻撃 : 6
   モンスターの攻撃 : 2
   やった! HPが6増えた!
   残りHP : 16
   あなたの攻撃 : 6
   モンスターの攻撃 : 4
   やった! HPが6増えた!
   残りHP : 22
   あなたの攻撃 : 1
   モンスターの攻撃 : 1
   やられた! HPが1減ってしまった
   HPはまだ21だ! 生き残ったあなたの勝ち!

```



プログラムをタイプするのが面倒な方へ： 前ページの「練習」コピペ用ページ

```
import random

HP = 10 # 勇者の HP
print(f"あなたは勇者です。HPは{HP}")
print("モンスターがあらわれた！")

for i in range(3): # 3回勝負
    print(f"残りHP {HP}")
    player = random.randint(1, 6)
    monster = random.randint(1, 6)

    print(f"あなたの攻撃 {player}")
    print(f"モンスターの攻撃 {monster}")

    if player > monster: # あなたの攻撃が強かった
        print(f"やった！HPが{player}増えた！")
        HP = HP + player
    else: # モンスターの攻撃が強かった
        print(f"やられた！HPが{monster}減ってしまった")
        HP = HP - monster

if HP < 0:
    print("3回勝負の結果、HPが残ってない...あなたの負けだ")
else:
    print(f"HPはまだ{HP}だ！生き残ったあなたの勝ち！")
```

必ず3回戦うってヘンだな…
普通HPが0以下になったら
そこで終わるのでは？

あ！先ほどのbreak使えば、
HPが0以下になったタイミングで
戦い終了できるのでは？



ゲーム3: どちらかが倒れるまで戦う ロールプレイング風ゲーム

```

▶ import random

HP = 10 # 勇者の HP
print(f"あなたは勇者です。HPは{HP}")

MonsterHP = random.randint(1, 10)
print(f"モンスターがあらわれた！ HPは{MonsterHP}")

for i in range(10000): # 非常に大きくしておく
    print(f"あなたの残りHP {HP}")
    print(f"モンスターの残りHP {MonsterHP}")
    player = random.randint(1, 6)
    monster = random.randint(1, 6)

    print(f"あなたの攻撃 {player}")
    MonsterHP = MonsterHP - player
    if MonsterHP <= 0:
        print("モンスターを倒した！")
        break

    print(f"モンスターの攻撃 {monster}")
    HP = HP - monster
    if HP <= 0:
        print("あなたは負けてしまった！")
        break

print("戦いが終わった！")

```

モンスターの登場時のHPを乱数で決める

あなたの攻撃でモンスターの
HPが減る。0以下になったら終了

モンスターの攻撃であなたの
HPが減る。0以下になったら終了

勝負がつくまで
ほぼ無限に
(10000回)
繰り返す



練習：どちらかが倒れるまで戦う ロールプレイング風ゲーム

```

import random

HP = 10 # 勇者の HP
print(f"あなたは勇者です。HPは {HP}")

MonsterHP = random.randint(1, 10)
print(f"モンスターがあらわれた！ HPは {MonsterHP}")

for i in range(10000): # 非常に大きくしておく
    print(f"あなたの残りHP {HP}")
    print(f"モンスターの残りHP {MonsterHP}")
    player = random.randint(1, 6)
    monster = random.randint(1, 6)

    print(f"あなたの攻撃 {player}")
    MonsterHP = MonsterHP - player
    if MonsterHP <= 0:
        print("モンスターを倒した！")
        break

    print(f"モンスターの攻撃 {monster}")
    HP = HP - monster
    if HP <= 0:
        print("あなたは負けてしまった！")
        break

print("戦いが終わった！")

```

自分で書いて
動かしてみよう！

ほぼ無限に近い戦い回数を設定

実行例

```

あなたは勇者です。HPは10
モンスターがあらわれた！ HPは3
あなたの残りHP 10
モンスターの残りHP 3
あなたの攻撃 1
モンスターの攻撃 6
あなたの残りHP 4
モンスターの残りHP 2
あなたの攻撃 3
モンスターを倒した！
戦いが終わった！

```

breakにより10000回を
待たずに途中終了

breakにより10000回を
待たずに途中終了



プログラムをタイプするのが面倒な方へ： 前ページの「練習」コピペ用ページ

```
import random

HP = 10 # 勇者の HP
print(f"あなたは勇者です。HPは{HP}")

MonsterHP = random.randint(1, 10)
print(f"モンスターがあらわれた！ HPは{MonsterHP}")

for i in range(10000): # 非常に大きくしておく
    print(f"あなたの残りHP {HP}")
    print(f"モンスターの残りHP {MonsterHP}")
    player = random.randint(1, 6)
    monster = random.randint(1, 6)

    print(f"あなたの攻撃 {player}")
    MonsterHP = MonsterHP - player
    if MonsterHP <= 0:
        print("モンスターを倒した！")
        break

    print(f"モンスターの攻撃 {monster}")
    HP = HP - monster
    if HP <= 0:
        print("あなたは負けてしまった！")
        break

print("戦いが終わった！")
```

いろいろできるよう
なってきましたねー！

自主練習：「どちらかが倒れるまで戦う ロールプレイング風ゲーム」の拡張案



- アイディア① モンスターの攻撃が6だったら「痛恨の一撃」と表示
 - `if monster == 6:` なら「痛恨の一撃」をprint
- アイディア② 「逃げる」か「戦う」かを毎回選べるようにする
 - for文の中で `input`を使う。「逃げる」を選択した時点で `break`
- アイディア③ 時々モンスターがHPを回復する
 - 例えば `monster == 5`の時だけ, MonsterHPを5倍する
- アイディア④ モンスターと何度も出会えるようにする
 - 今のfor文全体を, 別のfor文の中に入れる.
 - 毎回終了後, 負けてなければ勇者のHPを回復させてあげる...

自主練習：「どちらかが倒れるまで戦う ロールプレイング風ゲーム」の拡張案



- 前ページのアイデア以外にも，様々な拡張ができます！！
 - 皆さんは，（グラフィックを除き）本格的なRPGすら作れる知識をすでに学んでいます！
- Printを駆使して，物語風にしたり…
- ランダムに勇者やモンスターが攻撃を防御できたり…
- 勇者のレベルを入れて，勝利回数でレベルを上げたり，
 - さらに，レベルごとに攻撃力や防御力を変えたり…
- 勇者の攻撃のタイプを選べたり…
- MPを入れて，魔法攻撃もできるようにしたり…
- 様々なタイプのモンスターを登場させたり…
- などなど．プログラムとしては長くなりますが，
これまでの道具ですべてできます
 - 後述の「関数」を使えば，より作りやすくなります

プログラムはある意味
ブロックおもちゃと同じ！
シンプルな部品でも
どんどん組み合わせれば
超大作だって作れる！

