memo

プログラミング演習 I

授業開始までしばらくお待ちください。

プログラミング演習 I
Exercise on Programming I

本スライドは Teams で配布しています。

小林裕之・瀬尾昌孝
prog1@oitech.ddns.net

大阪工業大学 RD 学部システムデザイン工学科
OSAKA INSTITUTE OF TECHNOLOGY

6 of 14

a LaTeX + Beamer slideshow

授業前にこれを見ている人へ。

- 内容を先に読んで予習しておくことは大いに結構なことだと思います。ぜひ予習してください。
- 内容は随時更新しています。授業前に再度最新版を確認してください。
- 課題の提出 (Google Forms によるもの) は **授業当日の指示があるまで行わないように** してください。

復習の練習 1 難易度: ★★☆☆☆ (次問と合わせて 10 分)

有名な問題です、いちおう。

整数値を一つ入力し、それが

- 3 で割り切れれば “**`Fizz`**”、
- 5 で割り切れれば “**`Buzz`**”、
- 両方で割り切れば “**`Fizz Buzz`**”

と出力するプログラム**`fizzbuzz.py`**を作成せよ。

ヒント: 割り切れるかどうかは『あまり』を求める演算子 “`%`” を使うのが楽。

復習の練習 2 難易度: ★★★★☆ (前問と合わせて 10 分)

単純そうでけっこう難しい。

整数値を 3 つ入力すると、その中央値を出力するプログラム**median.py**を作成せよ。これまでに習ったことのみを使って実装すること。

```
someone@byod:~/prog1$ python3 median.py
3    # 1 つ目の入力
1    # 2 つ目の入力
4    # 3 つ目の入力
3    # 出力
```

動作確認として (1, 2, 3) (1, 3, 2) (2, 1, 3) (2, 3, 1) (3, 1, 2) (3, 2, 1) の 6 つの組み合わせを試してすべて 2 と出ればたぶん ok。

解答例

fizzbuzz.py(基本版)

```
a = int(input())
if a % 15 == 0:
    print("Fizz Buzz")
else:
    if a % 3 == 0:
        print("Fizz")
    if a % 5 == 0:
        print("Buzz")
```

fizzbuzz.py (**elif**版)

```
a = int(input())
if a % 15 == 0:
    print("Fizz Buzz")
elif a % 3 == 0:
    print("Fizz")
elif a % 5 == 0:
    print("Buzz")
```

median.py

```
x = int(input())
y = int(input())
z = int(input())
if x <= y and y <= z:
    print(y)
elif x <= z and z <= y:
    print(z)
elif y <= x and x <= z:
    print(x)
elif y <= z and z <= x:
    print(z)
elif z <= x and x <= y:
    print(x)
else:
    print(y)
```

※**median.py**では **if x <= y <= z:** のような書き方も可能ですが推奨しません。

コラム: elifって英単語的に意味不明

葛藤があったのでしょう。

elseとifが連続する『でなければ、もし』というのはプログラミングでは頻出するパタンです。C 言語やそれに近い文法の言語 (Java 等) の場合、ifの直後に直接一文書け、ブロックは波括弧で明確に示せるので、シンプルにelse ifと書くだけです。わかりやすい。ところが Python や Ruby は (理由はそれぞれで違いますが) 文法的にそれができません。そこで**苦肉の策**として else ifに相当するなにか別の予約語が必要になります。で、結局このようになりました。

- 案 1“elseif”: 長すぎ → 不採用
- 案 2“elsif”: else if と発音できなくもなくわかりやすい →Ruby が採用
- 案 3“elif”: 短い →Python が採用

```c
if (score >= 90) {
  printf("excellent!");
} else if (score < 60) {
  printf("failed.");
}
```

```ruby
if score >= 90
  puts "excellent!"
elsif score < 60
  puts "failed."
end
```

```python
if score >= 90:
  print("excellent!")
elif score < 60:
  print("failed.")
```

プログラミングの勉強をはじめたとき

Python ができるようになる まで繰り返す

予習をする

授業を受ける

復習をする

反復が大切ニャ。

元ネタ知ってます?

繰り返し処理に重要なもの２つ

ず〜っと

一般に、繰り返し処理には**繰り返す内容**と**繰り返す条件**がある。

繰り返し処理に重要なもの２つ

ず〜っと

基本料金を毎月 1000 円割引!

一般に、繰り返し処理には**繰り返す内容**と**繰り返す条件**がある。

繰り返し処理に重要なもの２つ

新規ご契約から一年間

ず〜っと

基本料金を毎月 1000 円割引!

一般に、繰り返し処理には**繰り返す内容**と**繰り返す条件**がある。

Python における繰り返しの基本のキ: `while`

`if` と似てるよ！

`if` **条件式** (など) `:`

インデント

条件式が成り立った時に実行する部分

一定数のスペース

`while` `:`

Python における繰り返しの基本のキ: `while`

`if` と似てるよ！

`if` **条件式** (など) :

インデント

条件式が成り立った時に実行する部分

一定数のスペース

`while` **条件式** (など) :

Python における繰り返しの基本のキ: while

if と似てるよ！

if **条件式** (など) **:**

インデント

条件式が成り立った時に実行する部分

一定数のスペース

while **条件式** (など) **:**

インデント

条件式が成り立って**いる間ず〜っと**実行する部分

一定数のスペース

Python における繰り返しの基本のキ: `while`

`if` と似てるよ！

`if` **条件式** (など) **：**

インデント

条件式が成り立った時に実行する部分

一定数のスペース

`while` **条件式** (など) **：**

インデント

条件式が成り立って**いる間ず〜っと**実行する部分

一定数のスペース

似ているよ、というか、まんま同じですね。

コラム: Python における `while` のあなたの知らなかった方が良かった (とも言い切れない) 世界

文法が **`if`** に似ていると聞いて反射的に頭に浮かぶのは『**`else`** や **`elif`** に相当するものはあるの?』という至極ごもっともな疑問です。ですが冷静に考えると、論理的に**`else`**や**`elif`**に相当するものは不要 (存在は可能だけど、論理的に冗長で無意味) ですよね? (わかる?) ですから多くの言語では **`if` に `else` はあっても `while` にはない**という文法を採用しています。

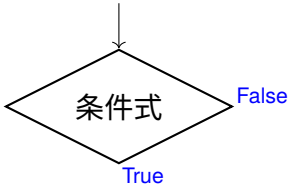
と・こ・ろ・が、あるんです、Python には**`while`**の相方の**`else`**が。これが一体何なんだと言うと、実は**`break`**でループを抜ける場合の挙動の違いを記述するものなのです。うーん、役に立つんだか立たないんだか。多重比較 (**`0 < m < 13`**みたいなの) 同様、あまり標準的でない文法なので使わないに如くは無し、ということで **`while`〜`else` とは距離を置いておくのが無難** だと思います。

あ、**`break`** については今回はやりません。ループを中断する方法でこちらはとても一般的なもので上手に使えばキレイなプログラムが書ける便利ツールなので近日中にこの授業でやります。

「〜の間ず〜っと繰り返す」の動作の詳細

わかっているようでわかっていないかも？

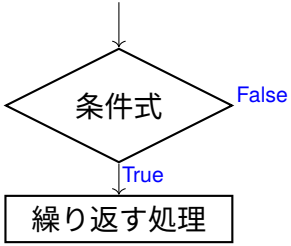
```
while 現在時刻 < 7時:
    布団をかけ直す
    枕の位置を直す
    目を閉じる
    5分寝る
起きる
身支度する
```



「〜の間ず〜っと繰り返す」の動作の詳細

わかっているようでわかっていないかも？

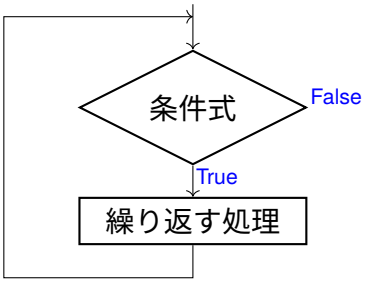
```
while 現在時刻 < 7時:
    布団をかけ直す
    枕の位置を直す
    目を閉じる
    5分寝る
起きる
身支度する
```



「〜の間ず〜っと繰り返す」の動作の詳細

わかっているようでわかっていないかも？

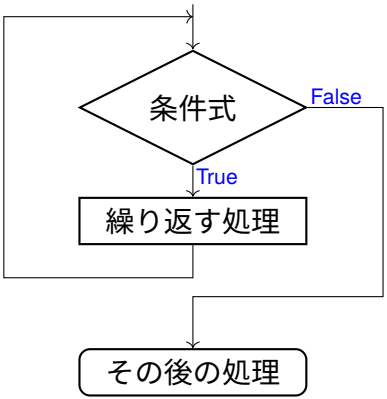
```
while 現在時刻 < 7時:
    布団をかけ直す
    枕の位置を直す
    目を閉じる
    5分寝る
起きる
身支度する
```



「〜の間ず〜っと繰り返す」の動作の詳細

わかっているようでわかっていないかも？

```
while 現在時刻 < 7時:
    布団をかけ直す
    枕の位置を直す
    目を閉じる
    5分寝る
起きる
身支度する
```



まずは簡単な例を見てみよう。

このあとこのプログラムをいじって遊ぶので全員入力してください。

loo.py (なんてファイル名だ……。)

```
i = 0
while i < 10:
    print(i * 3)
    i = i + 1
```

何が出力されるか**入力しながら**考えよう。(2 分)

よくある間違い

```
# something is lost

while i < 10:
    print(i * 3)
    i = i + 1
```

```
# something is lost
i = 0
while i < 10:
    print(i * 3)

```

それぞれどうなるかな?

いちばんよく使う定番の繰り返しの書き方

このパタンはもう丸暗記しよう

- カウンタとなる変数 (`i`が人気) を**初期化**
- **`while`**
- 繰り返したい処理本体
- カウンタを**インクリメント**

インクリメント: 1 増やす、という意味の業界用語。【対】デクリメント

いちばんよく使う定番の繰り返しの書き方

このパタンはもう丸暗記しよう

- カウンタとなる変数 (**i**が人気) を**初期化**
- **while**
- 繰り返したい処理本体
- カウンタを**インクリメント**

インクリメント: 1 増やす、という意味の業界用語。【対】デクリメント

```
# ここまでは繰り
# 返しと無関係
i = 0
while i < 42:
    # 繰り返したい
    # 処理をここに
    # 書く。
    i = i + 1
# ここからは繰り
# 返しと無関係
```

`while` ループのこの上なく単純な練習

このパタンをまずはとにかく丸暗記

練習 (3 分)

1 から 50 までの数を順番に出力するプログラム`fzbz.py`を作成せよ。

★☆☆☆☆

実行例:

```
1
2
3
（中略）
48
49
50
```

while ループのこの上なく単純な練習

このパタンをまずはとにかく丸暗記

練習 (3 分)

1 から 50 までの数を順番に出力するプログラム **fzbz.py** を作成せよ。

★☆☆☆☆

```
# fzbz.py
i = 1
while i <= 50:
    print(i)
    i = i + 1
```

実行例:

```
1
2
3
（中略）
48
49
50
```

本物の Fizz Buzz にチャレンジ

練習 (4 分)

fzbz.pyを改変して、1 から 50 までの数について、

- まずその数を出力し、
- 3 で割り切れれば “**Fizz**”、
- 5 で割り切れれば “**Buzz**”、
- 両方で割り切れば “**Fizz Buzz**”

と出力するプログラムを作成せよ。★☆☆☆

実行例:

```
1
2
3
Fizz
4
5
Buzz
（中略）
15
Fizz Buzz
16
（以下省略）
```

本物の Fizz Buzz にチャレンジ

練習 (4 分)

fzbz.pyを改変して、1 から 50 までの数について、

- まずその数を出力し、
- 3 で割り切れれば “**Fizz**”、
- 5 で割り切れれば “**Buzz**”、
- 両方で割り切れば “**Fizz Buzz**”

と出力するプログラムを作成せよ。★☆☆☆

```python
# fzbz.py (elif なし)
i = 1
while i <= 50:
  print(i)
  if i % 15 == 0:
    print("Fizz Buzz")
  else:
    if i % 3 == 0:
      print("Fizz")
    if i % 5 == 0:
      print("Buzz")
  i = i + 1
```

```python
# fzbz.py (elif 使用)
i = 1
while i <= 50:
  print(i)
  if i % 15 == 0:
    print("Fizz Buzz")
  elif i % 3 == 0:
    print("Fizz")
  elif i % 5 == 0:
    print("Buzz")
  i = i + 1
```

実行例:

```
1
2
3
Fizz
4
5
Buzz
（中略）
15
Fizz Buzz
16
（以下省略）
```

うるう年を列挙

練習 (7 分)

- 1896 年から 2112 年までのすべてのうるう年を列挙するプログラム **leaps.py** を作成せよ。
  ★★☆☆☆
- それらの合計を求めて、最後に出力せよ。★★★☆☆

実行例:

```
1896
1904
1908
（中略）
1996
2000
（中略）
2096
2104
2108
2112
106220
```

うるう年を列挙

練習 (7 分)

- 1896 年から 2112 年までのすべてのうるう年を列挙するプログラム **leaps.py** を作成せよ。
  ★★☆☆☆
- それらの合計を求めて、最後に出力せよ。★★★☆☆

```
# leaps.py
y = 1896
s = 0
while y <= 2112:
  if y % 400 == 0 or y % 4 == 0 and y % 100 != 0:
    s = s + y
    print(y)
  y = y + 1
print(s)
```

実行例:

```
1896
1904
1908
（中略）
1996
2000
（中略）
2096
2104
2108
2112
106220
```

入力でよくあるパターン

練習 (5 分)

endと入力されるまで繰り返し文字列入力を受け続け、最後に入力したすべての文字列をアンダースコアでつなげて出力するプログラム**snakecat.py**を作成せよ。★★★★☆

実行例:

```
user@host:~/prog1$ python3 snakecat.py
All                 # <- 入力 1 つめ
good                # <- 入力 2 つめ
things              # <- 入力 3 つめ
come                # <- 入力 4 つめ
to                  # <- 入力 5 つめ
an                  # <- 入力 6 つめ
end                 # <- 入力 7 つめ (end を入力)
All_good_things_come_to_an_end # -> 出力
user@host:~/prog1$
```

※次回やる**break**を使うときれいに実装できるけど今日やったことだけでも作れます。

入力でよくあるパターン

練習 (5 分)

endと入力されるまで繰り返し文字列入力を受け続け、最後に入力したすべての文字列をアンダースコアでつなげて出力するプログラム**snakecat.py**を作成せよ。★★★★☆

```python
# snakecat.py
a = input()
s = a
while a != "end":
    a = input()
    s = s + "_" + a
print(s)
```

実行例:

```
user@host:~/prog1$ python3 snakecat.py
All                 # <- 入力 1 つめ
good                # <- 入力 2 つめ
things              # <- 入力 3 つめ
come                # <- 入力 4 つめ
to                  # <- 入力 5 つめ
an                  # <- 入力 6 つめ
end                 # <- 入力 7 つめ (end を入力)
All_good_things_come_to_an_end # -> 出力
user@host:~/prog1$
```

※次回やる**break**を使うときれいに実装できるけど今日やったことだけでも作れます。

休憩 (ミニゲーム)

これまで学んだこと ** だけ ** で作ったゲームで遊ぼう！

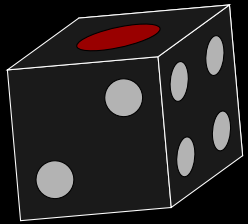
’80 年代初頭にはリアルにこんなゲームがたくさんありました。

1. 端末から以下を実行する。

```
git clone https://github.com/yagshi/avgpy
```

2. “command not found” などのエラーが出たら以下を実行して再度 1. を実行。
   - Windows: `sudo apt update; sudo apt install git`
   - macOS: `brew install git`
3. `cd avgpy`としてフォルダを移動し、`python3 game.py`として実行。
4. 途中でやめるときは [Control]+[C]。
5. 終わったら`cd ..` (cd, スペース, ドット, ドット) として元のフォルダに戻る。
6. せっかくなので VSCode からソースを見てみよう。

乱数



サイコロプログラム

dice.py

```
import random                  # これを冒頭に入れる
r = random.randint(1, 6)  #1〜6の乱数をrに代入
print(r)
```

乱数の作り方

random モジュール

random.randint 関数

乱数の作り方

random モジュール

import random とすると random モジュール (拡張機能のようなもの) が使えるようになる。……のだが、細かいことは今は気にせず、**乱数を使いたければプログラムの冒頭に import random と書く**と覚えておこう。

random.randint 関数

乱数の作り方

random モジュール

import random とすると random モジュール (拡張機能のようなもの) が使えるようになる。……のだが、細かいことは今は気にせず、**乱数を使いたければプログラムの冒頭に import random と書く**と覚えておこう。

random.randint 関数

random.randint(a, b) とすると、**a から b の範囲のランダムな整数**が得られる。

見せてもらおうか、`random.randint` 関数の性能とやらを。

練習 (5 分)

サイコロを 1,000,000 回振って、その平均値を求めるプログラム **`testrnd.py`** を作成せよ。(ホンモノのよくできたサイコロなら当然期待値は 3.5 だが、**`random.randint`**はどうだろう ?)

コラム: 乱数の “性能”

前のページの問題で見せてもらった性能は無作為性の一端に過ぎなかった。

真の乱数には性能は 3 つの性質があります。コンピュータが作り出す乱数は通常**疑似乱数 (pseudo random number)** というもので、本当の意味での乱数ではありません。多くの簡易疑似乱数は、以下のうち無作為性のみを実現しています。

- **無作為性** (randomness)
  値に統計的な偏りがないこと。
- **予測不可能性** (unpredictability)
  アルゴリズムが既知という条件下でも過去の系列から次の値を全く予測できないこと。
- **再現不可能性** (irreproductibity)
  同じ系列を再現しないこと。

(参考文献) 結城 浩 著「暗号技術入門」SB クリエイティブ株式会社

乱数で作る数当てゲーム

全く面白くないけど

練習 (5 分)

以下の動作をするプログラム**gtn.py**を作成せよ。

- ランダムで 1〜10 の数を作る。
- プレイヤーが数を入力。
- 正解だったら「**Bingo!**」、ハズレだったら「**Miss**」と出力。
- 終了。

実行例 (正解の場合):

```
hkoba@host:~/prog1$ python3 gtn.py
8
Bingo!
hkoba@host:~/prog1$
```

一応遊べる数当てゲーム: `gtn.py` ver. 2.0

練習 (5 分?) (難易度: ★★★★☆)

- ランダムで 1〜10 の数を作る。
- プレイヤーに数を入力させる。
- 正解だったら「**`Bingo!`**」、ハズレだったら「**`Miss`**」と出力する。
- 正解だったら終了、ハズレだったら**入力に戻る**。

終わったら次ページの追加アップデートを行おう。

少しは楽しめる数当てゲーム: gtn.py ver. 3.0

前ページの練習ができて時間を持て余している人向け

改良点 (難易度: ★☆☆☆☆)

ハズレのときに**Miss**だけでなく、ヒントもだそう。

- 答えのほうが大きければ
  “**The answer is larger.**”
- 答えのほうが小さければ
  “**The answer is smaller.**”

ヒントがあるので数の範囲も 1〜100 にしよう。

Guess The Number ver. 1.0, 2.0, and 3.0

```python
# ver 1.0
import random
r = random.randint(1, 10)
x = int(input())
if x == r:
    print("Bingo!")
else:
    print("Miss")
```

Guess The Number ver. 1.0, 2.0, and 3.0

```python
# ver 1.0
import random
r = random.randint(1, 10)
x = int(input())
if x == r:
    print("Bingo!")
else:
    print("Miss")
```

```python
# ver 2.0
import random
r = random.randint(1, 10)
x = int(input())
while x != r:
    print("Miss")
    x = int(input())
print("Bingo!")
```

```python
# ver 3.0
import random
r = random.randint(1, 100)
x = int(input())
while x != r:
  print("Miss")
  if x > r:
    print("The answer is smaller.")
  else:
    print("The answer is larger.")
  x = int(input())
print("Bingo!")
```

▲を描く

練習 (5 分) (難易度: ★★★☆☆)

整数値を入力するとその高さの山を出力するプログラム **mtfill.py** を作成せよ。(右の実行例参照)

実行例

```
8          <---  整数値を入力
           X
          XXX
         XXXXX
        XXXXXXX
       XXXXXXXXX
      XXXXXXXXXXX
     XXXXXXXXXXXXX
    XXXXXXXXXXXXXXX
```

△を描く

練習 (5 分) (難易度: ★★★★☆)

2 以上の整数値を入力するとその高さの山の枠を出力するプログラム **mtframe.py** を作成せよ。極力短くシンプルなプログラムを目指すこと。(右の実行例参照)

実行例

```
8        <--- 整数値を入力
        X
       X X
      X   X
     X     X
    X       X
   X         X
  X           X
 X             X
XXXXXXXXXXXXXXX
```

今回の復習にちょうどよい paiza 練習問題

家で自習しよう！

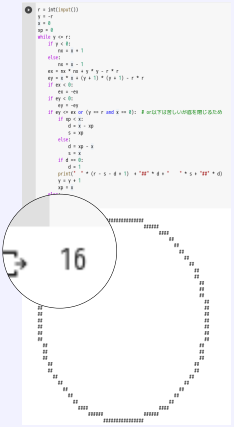
【paiza ラーニング】→ 【問題集】→ 【C ランク獲得】→ 【ループメニュー 1】→STEP 1, 2, 5, 6

○を描く

練習 (200 分超?) (難易度: ★★★★★+)

数値 (r) を入力すると横半径が $2r$、縦半径が r の円を描くプログラムを作成せよ。横半径が 2 倍なのは、半角文字の縦横比が 2:1 だからである。
右図は $r = 16$ のときの実行例。**これまでに習ったことのみ**で何とかなる。(アルゴリズムによって若干形状が変わる場合もあるので厳密に右図のとおりでなくとも半径がおよそ r でだいたい丸ければ良い。)

右図は $x^2 + y^2 = r^2$ を使った場合の解答例。拡大すれば答えはわかるが少なくとも 3 時間は見ないで考えよう！他にも $\frac{d}{d\theta}x(\theta) = y(\theta)$ を使ったり単振動の運動方程式的なものを使ったり、アルゴリズムはいくつか考えられると思う。



練習: 以下の仕様のプログラムを作成せよ。

自己チェカーの「箱・改」でチェックできます。

- **整数値**で幅 (例: 16)、高さ (例: 5) を入力し、最後に**1 文字の文字列**(例: B) を入力すると、指定幅・高さの角を落とした四角形を出力するプログラムを作成せよ。ただし、幅・高さとも 3 未満を入力したら 3 として扱うこと。
- 実行例は右。数値と文字によって結果がきちんと変わるように。
- **これまでの授業でやったことのみで実装せよ。**
- これまでの授業でやったことのみで 20 行未満で実装した正しく動作するプログラムは 3 点。20 行以上で正しく動作するプログラムは 2 点。それ以外は 0〜1 点。

```
user@host:~/prog1$ python3 kadai.py
16      <-- 数値（幅）を入力
5       <-- 数値（高さ）を入力
B       <-- 1 文字の文字列を入力
 BBBBBBBBBBBBBB
B              B
B              B
B              B
 BBBBBBBBBBBBBB
user@host:~/prog1$
```

課題: 以下の仕様のプログラムを作成せよ。

自己チェカーの「横山」でチェックできます。

- 整数値を入力すると**x**で描かれたその高さの**横倒しの黒山もしくは白山**(どちらか好きな方) を左に寄せて実行例のように出力するプログラムを作成せよ。
- 入力した数値が 1 未満の場合は 1 として処理すること。
- 実行例は下のとおり。左が黒山、右が白山。左右の向きに注意。
- 黒山なら 2 点、白山なら 3 点。**必ず自力でできるものをやること**。
- **習ったことのみ**を使って実装すること。習っていないものを使ったら (自動採点や締め切り前の成績フィードバックでは 3 点でも) **あとから減点** する。

```
user@host:~/prog1$ python3 kuroyama.py
4     <-- 数値（横方向の高さ）を入力
X
XX
XXX
XXXX
XXX
XX
X
user@host:~/prog1$
```

```
user@host:~/prog1$ python3 shiroyama.py
6     <-- 数値（横方向の高さ）を入力
     X
    XX
   X X
  X  X
 X   X
X    X
 X   X
  X  X
   X X
    XX
     X
user@host:~/prog1$
```

第 6 回 (5/20) 課題 (提出期間: 5/20〜5/26)

- 提出前に自己チェック。(学外からは**要 VPN**)
  `https://edu2.rd.oit.ac.jp:4000`
- 課題提出 Google Forms から提出。(VPN 不要)
  最後の 送信 クリックを忘れないこと！
- 提出できたら**確認 e-mail が届く**ので必ず確認。
- 期間外の提出は**自動的に** 別の課題を上書きするような処理をされてしまうので期間外の提出は厳禁。(早く提出するのも、遅れて提出するのもどっちもダメ。)
- 複数回提出した場合は**最後のもののみが有効**。

https://forms.gle/7Wy4CpDQ819MfwVx5