
Python超入門
RICORA言語班 資料

資料作成：情報科学科3年 越塚 育

講座の目的

- Python の簡単なプログラムが作れるようになる。
- コンピュータやプログラミングの基本的な知識を得る。
- プログラミングの楽しさを知る。

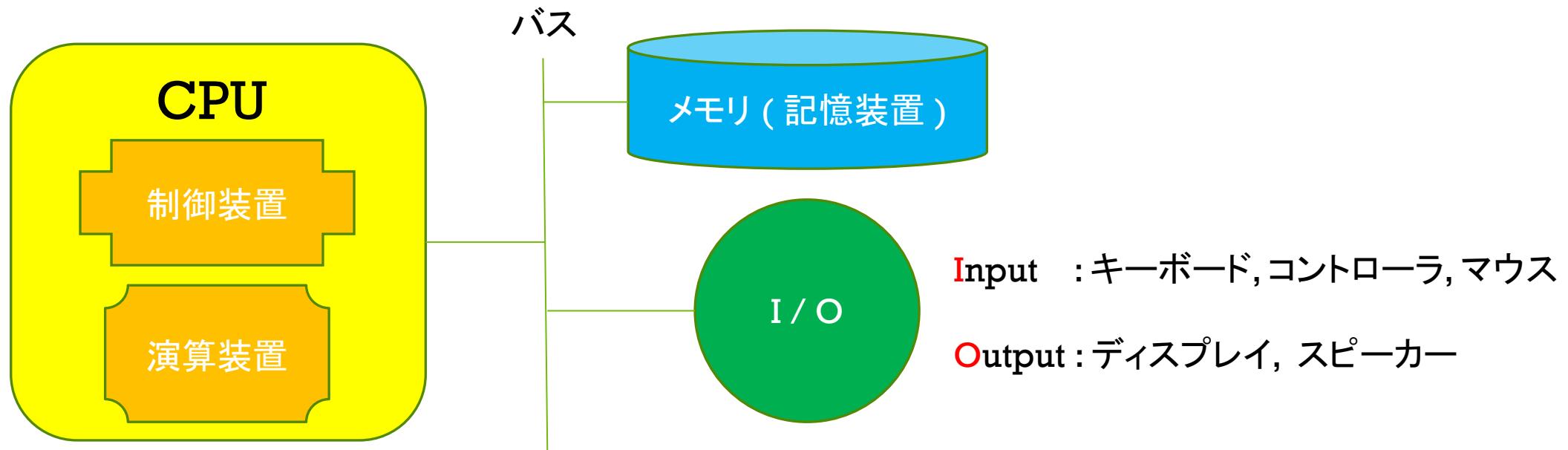


コンピュータとは？

- コンピュータ

計算処理を自動で行う機械

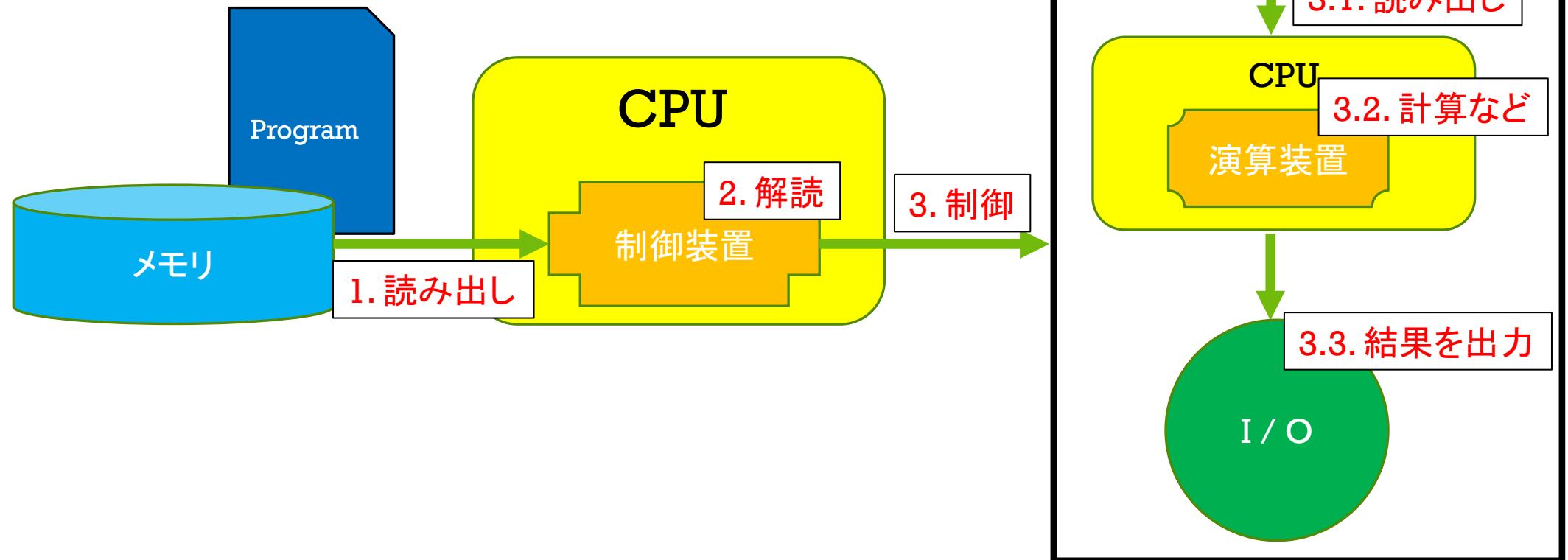
- コンピュータの仕組み





プログラムとは？

- プログラム
コンピュータに対する命令
- プログラムの実行



今回使う 開発環境

Paiza.io

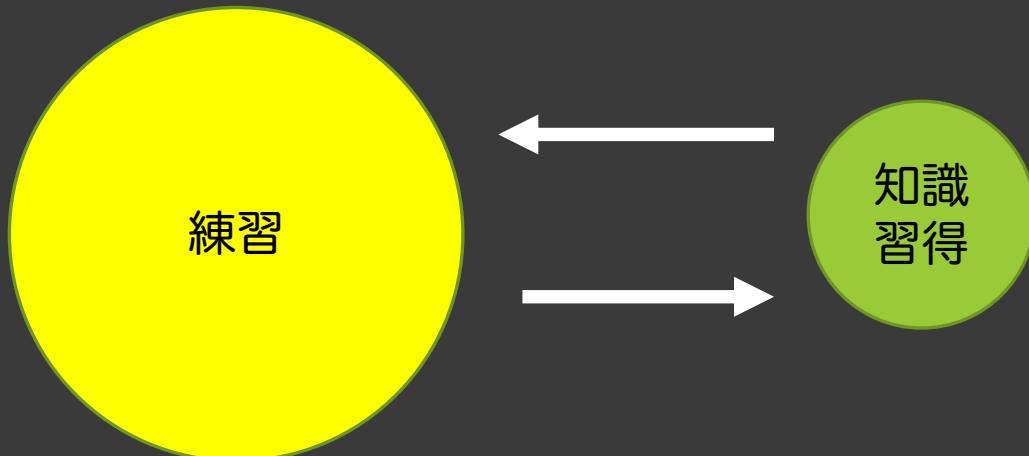
<https://paiza.io/ja>



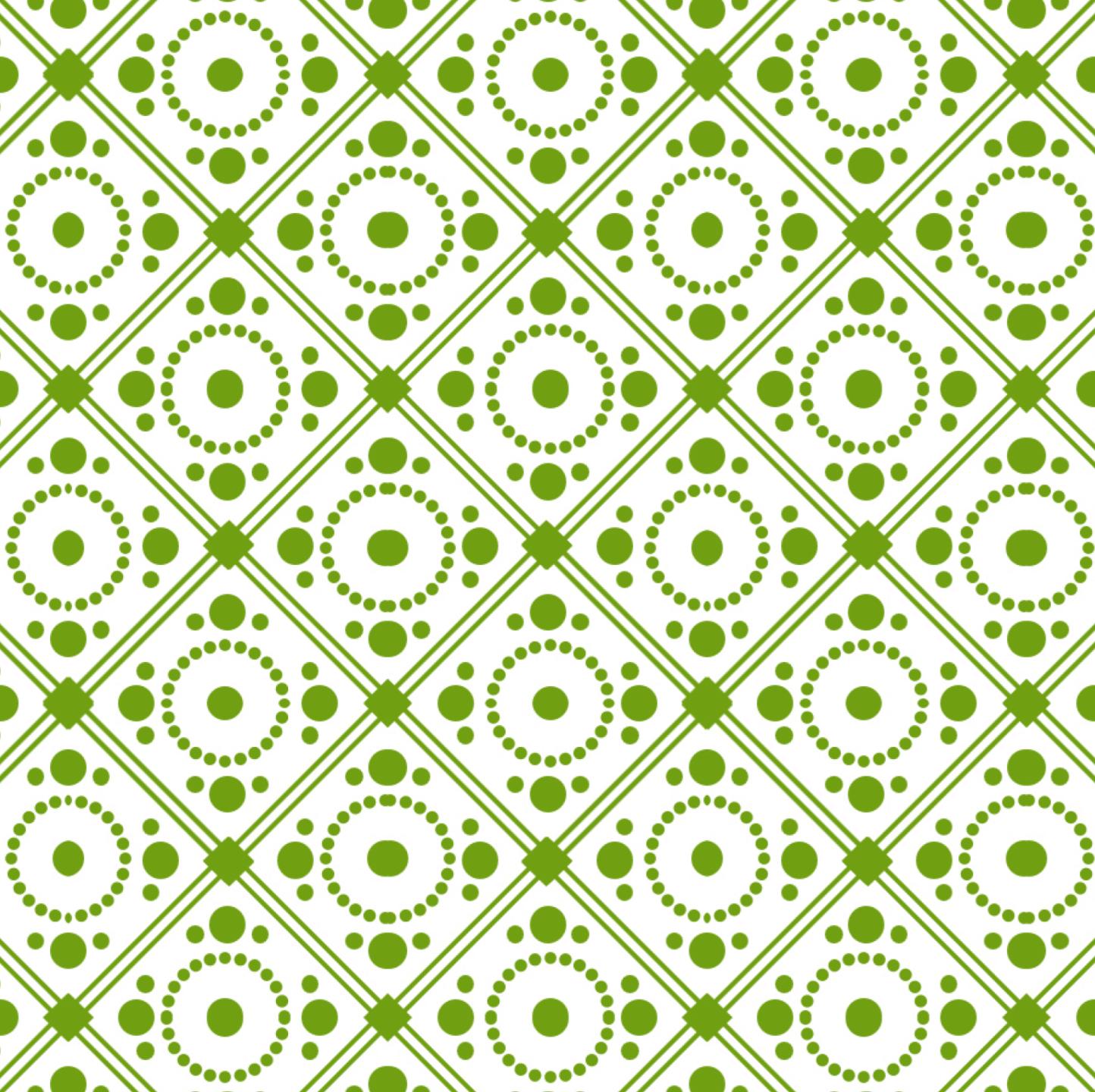
> paiza.IOはオンラインですぐにプログラミングが始められる、オンライン実行環境です。
C,C++,Java,Ruby,Python,PHP,Perlなど 主要24言語に対応。
ファイルアップ機能、外部apiへの接続や、スクレイピングなども可能です。

プログラミング学習の勧め

たくさんプログラムを書こう



プログラミング を始めよう



Hello World

まずは、次のプログラムを書いて実行してみましょう。

```
print("Hello World!!")
```

次に、わざと間違えてみましょう。

```
prin("Hello World!!")
```

実行結果

```
Traceback (most recent call last):
  File "Main.py", line 4, in <module>
    prin("Hello World!")
NameError: name 'prin' is not defined
```

prin なんて定義されてないよと怒れられていますね。

Pythonの基本ルール

1. 上から下へ 1行ずつ実行される。
2. 全角文字は基本使わないこと。特に全角スペース、ダメ絶対。
3. インデントを守る。

```
print("Hello World!!")
```

わざと先頭にスペースを空けて実行してみる

実行結果

```
Traceback (most recent call last):
  File "Main.py", line 4
    print("Hello World!")
IndentationError: unexpected indent
```

4. インデントが崩れたり、文法が崩れたりしなければ、
スペースと空行は無視される。

Okな例

```
print ( "Hello World!" )
```

Ngな例

```
pri nt("Hello World!")
```

基本的な文法

1. 予約語

プログラム言語で、あらかじめ意味の決まっている単語

予約語を全部表示するプログラム

```
print(__import__('keyword').kwlist)
```



※ 使っていくうちに覚えるものなので意識的に覚えなくて良い。

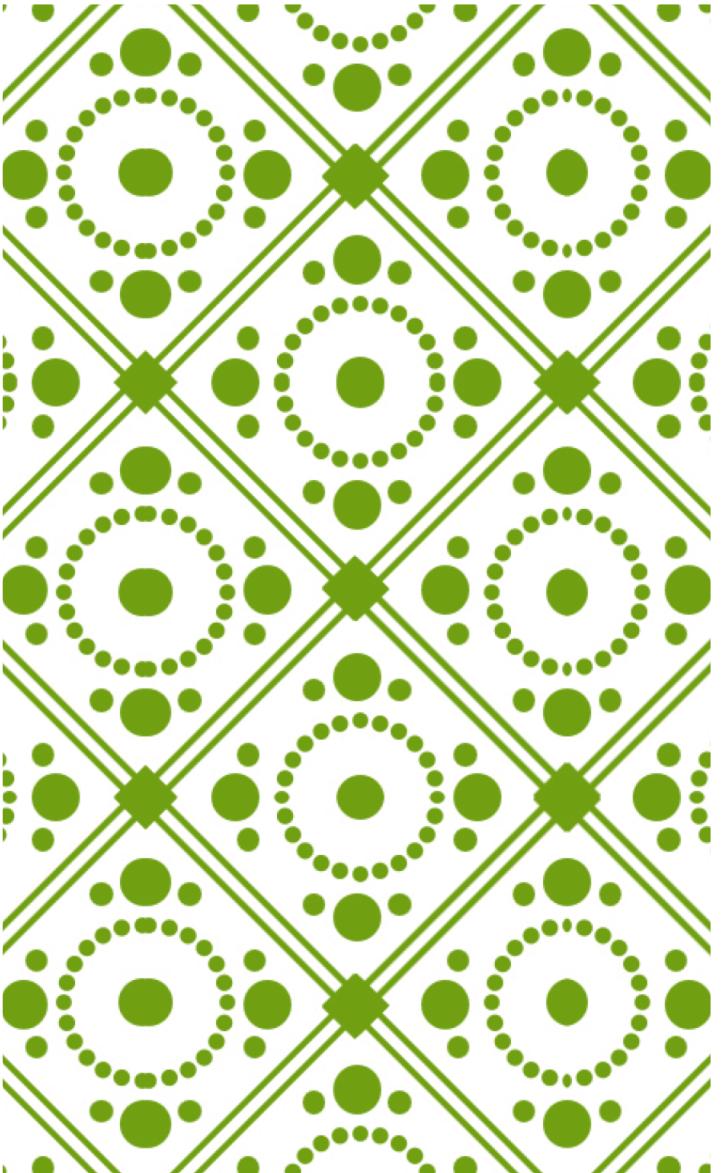
2. コメント

プログラムとしては認識されない記述

- プログラムを読みやすくするために、説明などを書いておく。
- 不要なプログラムを一時的に消す。

Pythonでは、#から行末まではコメントとして扱われる。

```
# Hello Worldの例題です。  
print("Hello World") # ここに書いててもok  
# prin("Hello World")
```



変数

計算結果や計算途中のデータを保存する方法

変数

■ 変数

値を保存するもの

- プログラム中で計算結果や
外部からの入力データなどを保持するために使う。

※ 数学の変数 (x とか y とか) とは別物での注意 !!

■ 変数の名前

アルファベット, 数字, アンダースコア (_) の3種類が使用可能。
スペース, 予約語, Built-in Function は使えない。

■ 変数の使い方

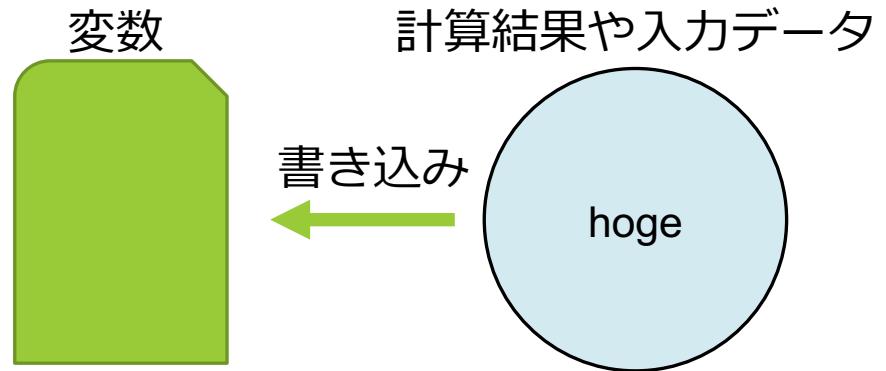
1. 代入

変数に値を保存すること。

[記述方法]

<変数> = <保存したい値>

※ = は数学のイコールとは別物!!



2. 参照

<変数> に保存されている値を使うこと。



Hello World2

変数を使って、世界とYouTubeに挨拶してみる。

helloWorld2.py

```
message = "Hello World!"  
print(message)
```

```
# messageに保存してあった "Hello World!" は上書きされる。  
message = "Boon Boon Hello Youtube!!"  
print(message)
```

実行結果

```
| Hello World!  
| Boon Boon Hello Youtube!!  
|
```

変数名の付け方

- プログラムを読むときに意図が伝わるような名前をつける。
- なるべく英語で名前をつける。ローマ字はなるべくやめよう。
- 2語以上の変数名をつけるときは、2通りの書き方がある。

(例) english message

english_message または englishMessage



データ型

プログラムにおけるデータの扱い方について学ぶ

String (文字列)

[記述方法] <String> ::= “ 文字 “ | ‘ 文字 ‘

シングルクウォート ‘ またはダブルクウォート “ のいずれかで
始まりと終わりを囲う。始まりと終わりは同じものを使うこと。

[例] “Hello World“, ‘RICORA’

エスケープシーケンス

キーボードで入力できない文字を文字列として書く方法。

バックスラッシュと別の文字を組み合わせて1つの文字を記述する。

\n : 改行 ,

\t : タブ,

\" : ダブルクオート

\' : シングルクオート

\\" : バックスラッシュ

escape.py

```
print("Hello \nWorld!!")
```

実行結果

```
"Hello  
World!!"
```

Stringに対する操作

1. 文字列の結合 <String> ::= <String> + <String>

演算子 + を使う。

concat.py

```
greeting = "Hello "
print(greeting + "World!")
print("Boon Boon " + greeting + "Youtube!")
```

実行結果

```
Hello World!
Boon Boon Hello Youtube!
```

2. 先頭からN文字目を参照する $\text{<String>} ::= \text{<String>}[\text{N}]$
文字列の後ろに [N] をつける。 (N は取り出したい文字の番目)

先頭の文字は0番目と数える。 (0-index)

(例) “ABCDEFG”

Aは0番目の文字, Bは1番目の文字

getSubstring.py

```
name = "Tokyo University of Science"
```

```
# 文字列の1番目の文字
```

```
print(name[1])
```

```
# 文字列の 0文字目, 6文字目, 20文字目を抜き出して結合.
```

```
abbreviation = name[0] + name[6] + name[20]
```

```
print("略称: " + abbreviation)
```

実行結果

O
略称: TUS

※ 存在しない番目の文字を参照しようとすると、エラーが起きる

outOfRange.py

```
string = "ABC"  
print(string[3])
```

実行結果

```
| Traceback (most recent call last):  
|   File "outOfRange.py", line 5,  
|     in <module>  
|         print(string[3])  
| IndexError: string index out of range
```

Number (数)

- └ int (整数) 整数を扱うデータ型
- └ float (小数) 小数を扱うデータ型

[記述方法]

普通に書けば、整数型や浮動小数点数型として扱われる。

`int` ⇌ `float` は割と空気を読んで勝手に変換してくれるので、
Pythonではそこまで意識しなくてok

Numberに対する操作

1. 算術演算子 <Number> ::= <Number> (算術演算子) <Number>

算術演算子	結果
+	左辺 + 右辺
-	左辺 - 右辺
*	左辺 × 右辺
/	左辺 / 右辺 (少数を返す)
%	左辺 mod 右辺
**	左辺 ^ 右辺
//	左辺 / 右辺 (商の小数部分を切り捨て)

計算の優先順位： 掛け算, 割り算 >> 足し算, 引き算

通常の数式同様、先に計算させたい部分には()をつける。

Number.py

```
x = 10  
y = 3  
z = 2.5  
  
print("x % y")  
print(x % y)  
  
print("x // y")  
print(x // y)  
  
print("x ** y")  
print(x ** y)  
  
print("x + y * z ")  
print( x + y * z )  
  
print("( x + y ) * z ")  
print( ( x + y ) * z )
```

実行結果

```
x % y  
1  
x // y  
3  
x ** y  
1000  
x + y * z  
17.5  
( x + y ) * z  
32.5
```

2. 比較演算子

代入操作と算術演算子を使った計算をまとめた操作

代入演算	結果
$a += b$	$a = a + b$
$a -= b$	$a = a - b$
$a *= b$	$a = a * b$
$a /= b$	$a = a / b$
$a \%= b$	$a = a \% b$
$a **= b$	$a = a ** b$
$a //= b$	$a = a // b$

assignment_operation.py

```
a = 1  
b = 3  
a += 2  
a *= b  
print(a)
```

実行結果

```
9
```

Boolean (論理)

真理値のTrue(真)とFalse(偽)という2値を扱うデータ型
主に条件文を記述する時に使う。(if 文等の制御構文)

[記述方法]

True, False と書けば、Boolean型の値となる。

Booleanに対する操作

1. 論理演算子 $\langle \text{Boolean} \rangle ::= \langle \text{Boolean} \rangle \text{ and } \langle \text{Boolean} \rangle \mid \langle \text{Boolean} \rangle \text{ or } \langle \text{Boolean} \rangle \mid \text{not } \langle \text{Boolean} \rangle$

論理演算子	結果
and	左辺 Trueかつ右辺 Trueの時のみTrue
or	左辺 Trueまたは右辺 Trueの時True
not	右辺がTrueならFalse, 右辺がFalseならTrue

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

Boolean型の値を返す操作

1. 比較演算子 <Boolean> ::= <Number> <比較演算子> <Number> | <String> <比較演算子> <String>

比較演算子	結果
<	左辺 < 右辺 なら true
<=	左辺 ≤ 右辺 なら true
>	左辺 > 右辺 なら true
>=	左辺 ≥ 右辺 なら true
==	左辺と右辺が等しいなら true
!=	左辺と右辺が異なるなら true

String型の大小比較は、**辞書順**（辞書に出てくる順）で行われる。

comp.py

```
print (1 < 2)
print (3 != 3)
print ( ( -2 < 3 ) and ( not ( 0 <= 1 ) ) )
print ("abc" == "abc")
print ("abc" <= "defg")
```

実行結果

```
True
False
False
True
True
```

int型, float型とString型を大小比較しようとするとエラーになる。

comp_int_string.py

```
print ( 1 < "32" )
```

実行結果

```
Traceback (most recent call last):
  File "comp_int_string.py", line 1, in <module>
    TypeError: '<' not supported between instances of 'int' and 'str'
```

型の変換

■ String型へ変換

[記述方法]

`str(<値>)`で変換する。

(例) `str(3)` → “3”, `str(3.14)` → “3.14”, `str(True)` → “True”

■ int型, float型へ変換

[記述方法]

`int(<値>), float(<値>)`で変換する。

(例) `int("1")` → 1, `float("1.3")` → 1.3, `int("abc")` → Error (できない)

`int(True)` → 1, `int(False)` → 0

リスト

関連データをひとまとめにして扱うデータ構造の1つ

リストのイメージ

title. Pythonのリストとは

0. 名前のついた箱の列
1. 箱には0から順に番号が付いている
(ついている番号を `index` という)
2. 箱1つ1つに値を保存する。
箱1つ1つが変数と考えられる。
3. 番号を指定することで中身を見れる
(例) `fruits[1]` → “みかん” (String)

Pythonのリスト
通常のリスト



リスト

- リスト
複数のデータをまとめて管理することができるデータ型
- リストの作り方
 - [<要素0>, <要素1>, <要素2>, ...]
 - ※ 各要素の型は異なっても良い。
- (例)

```
fruits = [ "apple" , "orange" , "banana" , "grape" , "pineapple" ]  
  
numbers = [ 1, 5, 7, -2, 1.3 , 10.1 ]  
  
mixList = [ "begin" , 1 , [ 10, 100] , 1000 , "end" ]  
  
empty = [ ] # 要素の入っていない空リストを作ることができる。
```

リストに対する操作

1. indexを指定して要素を参照する <List>[index]

リストの後ろに [index] をつける。

list_index.py

```
fruits = [ “apple” , “orange” , “banana” , “grape” , “pineapple” ]
print(fruits[1])

two_dim = [ [ 1,2 ] , [ 3,4 ] ]
print(two_dim[0])
print(two_dim[0][1])
```

実行結果

```
“orange”
[1,2]
2
```

2. リストの変更

Pythonのリストは動的に変更が可能

変更の種類

- 要素の変更
- 要素の追加
- 要素の削除

2.1 要素の変更 $<\text{List}>[\text{index}] = <\text{変更後の値}>$

代入文で変更を行う。

update_list.py

```
fruits = [ "apple" , "orange" , "banana" , "grape" ]
print(fruits)
fruits[2] = "melon"
print(fruits)
```

実行結果

```
[ "apple" , "orange" , "banana" , "grape" ]
[ "apple" , "orange" , "melon" , "grape" ]
```

2.2 要素の追加(末尾) <List>.append(<追加する値>)

リストの末尾に要素を追加する

append_list.py

```
fruits = [ "apple" , "orange" , "banana" ]  
print(fruits)  
fruits.append("melon")  
print(fruits)
```

実行結果

```
[ "apple" , "orange" , "banana" ]  
[ "apple" , "orange" , "banana" , "melon" ]
```

appendだけでリストを作ることもできる。

append_only_list.py

```
fruits = []  
fruits.append("apple")  
fruits.append("orange")  
fruits.append("banana")  
print(fruits)
```

実行結果

```
[ "apple" , "orange" , "banana" ]
```

2.3 要素の追加 (indexで指定) <List>.insert(<index>, <追加する値>)

リストのindexで指定した場所に要素を追加する。

insert_list.py

```
fruits = [ "apple" , "orange" , "banana" ]  
print(fruits)  
fruits.insert(1, "melon")  
print(fruits)
```

実行結果

```
[ "apple" , "orange" , "banana" ]  
[ "apple" , "melon" , "orange" , "banana" ]
```

2.4 要素の削除 (indexで指定) <List>.pop(<index>)

リストのindexで指定した場所の要素を削除し、 値を返す。

pop_list.py

```
fruits = [ "apple" , "orange" , "banana" ]  
deletedFruits = fruits.pop(1)  
print(fruits)  
print("deleted fruit : " + deletedFruits)  
fruits.pop() #末尾削除の時はindex省略可  
print(fruits)
```

実行結果

```
[ "apple" , "banana" ]  
deleted fruit : orange  
[ "apple" ]
```

2.5 要素の削除(要素で指定) <List>.remove(<削除する値>)

リストから指定した値を1つ削除する。

指定した値がリスト内に複数ある場合は、1番indexの小さいものを削除する。

remove_list.py

```
fruits = [ "apple" , "orange" , "banana" ]  
print(fruits)  
fruits.remove("apple")  
print(fruits)
```

実行結果

```
[ "apple" , "orange" , "banana" ]  
[ "orange" , "banana" ]
```

3. リストの結合 <List> ::= <List> + <List>

演算子 + を使う。

concat_list.py

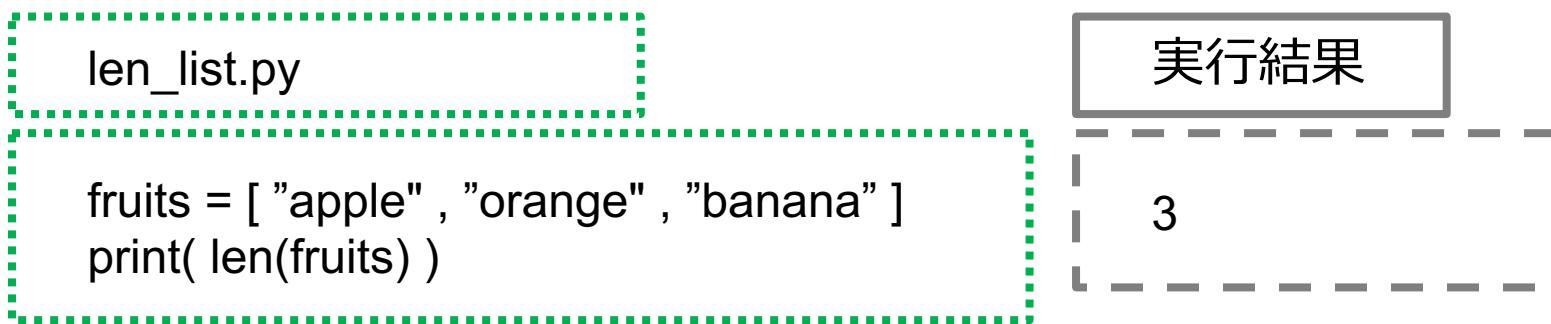
```
fruits1 = [ "apple" , "orange" , "banana" ]  
fruits2 = [ "grape" , "peach" ]  
fruits = fruits1 + fruits2  
print(fruits)
```

実行結果

```
[ "apple" , "orange" , "banana" , "grape" , "peach" ]
```

4. リストの長さを取得する `len(<List>)`

リストの要素数(長さ)を取得することができる。



5. リストの要素をソートする `<List>.sort()`

リストのデータをソートする。

★ データがある規則に従って並び変えることをソートという。

(例.)

数字の列を小さい順に並べ替える。

文字列の列を辞書順が大きい順に並び替える。

ソートにはいろいろな方法がある。

list_sort.py

```
numbers = [ 6, 1, 2, -3, 10, 2 ]  
numbers.sort()  
print(numbers)
```

実行結果

```
[ -3, 1, 2, 2, 6, 10 ]
```

[コラム] ソートアルゴリズム

■ ソートアルゴリズムを通して学べること

1. 計算量の改善 ($O(n^2)$ から $O(n \log n)$)
2. 再帰の考え方, 分割統治法 (クイックソート, マージソートなど)
3. ヒープやバケットなどのデータ構造
4. 乱拓アルゴリズムの思想

最低限知っておきたいソート：

バブルソート, 選択ソート,
マージソート, クイックソート, バケットソート

参考資料

最初に見ると良い動画（秋葉拓哉先生が教えてくれるよ！）

https://www.youtube.com/watch?v=o_Szfe9YbLQ

- バブルソート

<https://www.youtube.com/watch?v=IHFb0wYBRO>

- 選択ソート

<https://www.youtube.com/watch?v=A1UuoXNoy2w>

- マージソート（分割統治）

<https://www.youtube.com/watch?v=a3yXjNhGwt0>

- クイックソート（分割統治、乱拓）

<https://www.youtube.com/watch?v=a3yXjNhGwt0>

- バケットソート（バケット）

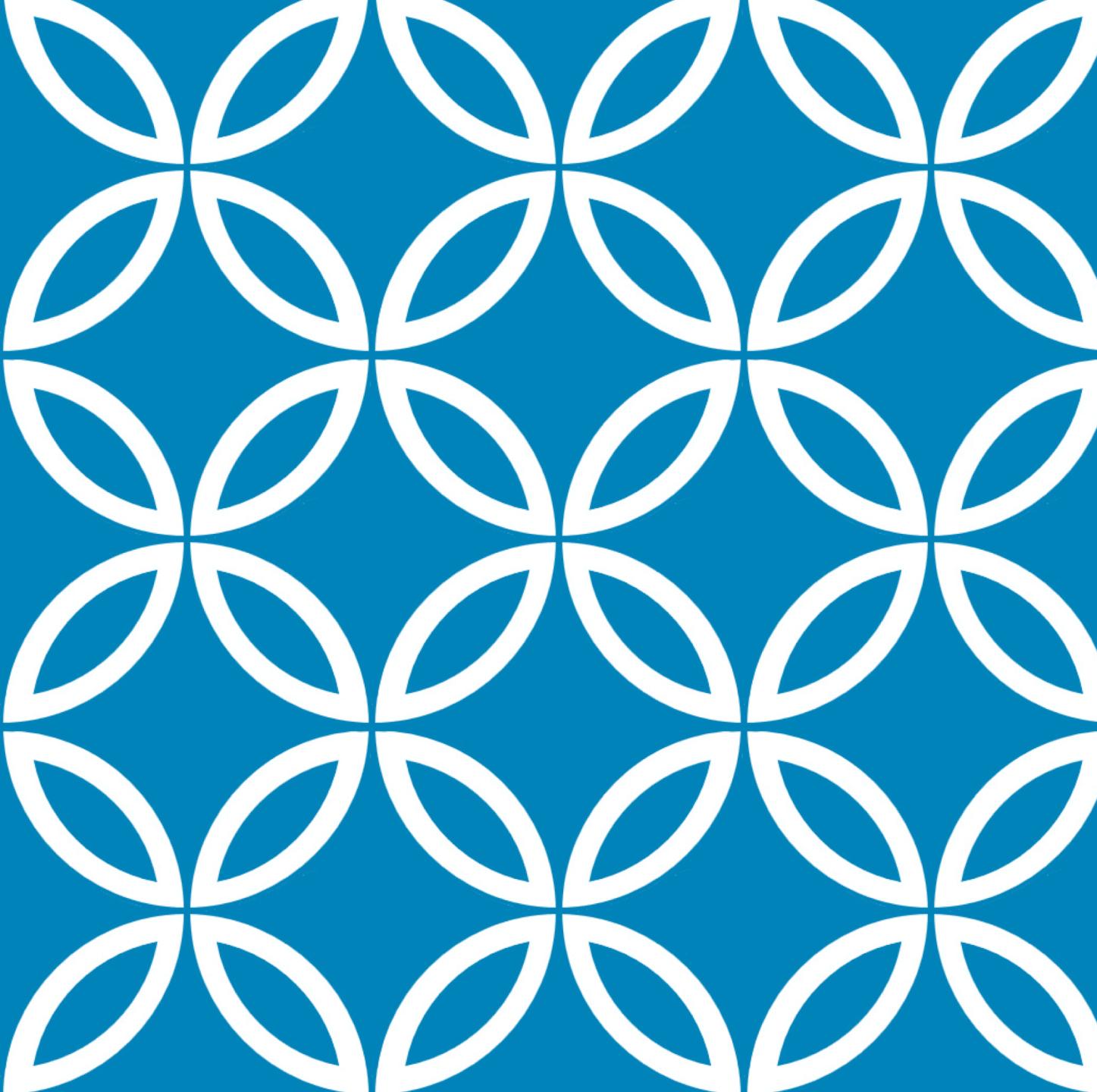
https://www.codereading.com/algo_and_ds/algo/bucket_sort.html

全体像を知りたい

<https://qiita.com/drken/items/44c60118ab3703f7727f>

制御構文 FOR

繰り返し処理の制御構文を学ぶ



リスト要素の繰り返し処理

[記述方法]

```
for <変数> in <List>:
```

■ ループ内の実行文

- ★ for文の中で実行するプログラムは必ずインデントする。
- ★ for文を終わるときは、インデントをしないようにする。

- リスト内の要素が先頭から順に変数へ代入される。
- Pythonでは、インデントを使って制御構文のブロックを表現する。

for_example.py

```
fruits = [ “apple” , “orange” , “banana” ]  
  
for fruit in fruits:  
    print(fruit)  
  
print(“the end of loop”)
```

実行結果

```
apple  
orange  
banana  
The end of loop
```

コレクション

複数のデータが集まったデータ型。
簡単にいうとリストっぽいもの。

(例) list, dict, tuple, set, range

ほとんどのコレクションは、
for <変数> in <Collection>:
 ループ内の実行文

として、複数のデータを処理できる。

rangeを使った繰り返し処理

■ range

整数専用のコレクション。

※ 厳密にはlist型ではない。list(<range>) とすればlist型に変換できる。

(例)

■ range型オブジェクトの作り方

range(n) : [0 , 1 , 2 , ... , n-1]

range(a, b) : [a, a+1, a+2, ... , b-1]

range(a, b, c): [a, a+c, a+2*c, ... ,]

※ 5は含まないので注意

range(5) = [0, 1, 2, 3, 4]

※ 3は含むけど7は含まないので注意!!

range(3, 7) = [3, 4, 5, 6]

※ 10は含まないので注意!!

range(1, 10, 3) = [1, 4, 7]

range_loop1.py

```
for i in range(3):
    print("Hello")
    print(i)
```

実行結果

```
Hello
0
Hello
1
Hello
2
```

range_loop2.py

```
sum_value = 0
for i in range( 4,20, 3):
    sum_value += i
print(sum_value)
```

実行結果

```
69
```

等差数列 $4, 7, 10, 13, 16, 19$ の和を
 $4 + 7 + 10 + 13 + 16 + 19$ を直接計算することで求めています。

練習問題 1

問題 forループを使って、 $n!$ を求めるプログラムを作れ

※ 入力の受け取り方

`<変数> = input()`

とすれば、入力が`<変数>`にString型として代入される。

整数を入力として受け取りたいときは、

`<変数> = int(input())`

ヒント1 `range(1, n+1)` を使おう。

ヒント2 `range_loop2.py` を参考にしてみよう。

解答例

factorial.py (解答例1)

```
n = int(input())
ans = 1
for i in range(1, n+1):
    ans *= i
print(ans)
```

factorial.py (解答例2)

```
n = int(input())
ans = 1
for i in range(n, 0, -1):
    ans *= i
print(ans)
```

[コラム] String型の繰り返し処理

実は String型も for文を作ることができる。

[記述方法]

```
for <変数> in <String>:  
    ループ内の実行文
```

```
string = "RICORA"  
for char in string:  
    print(char)
```

実行結果

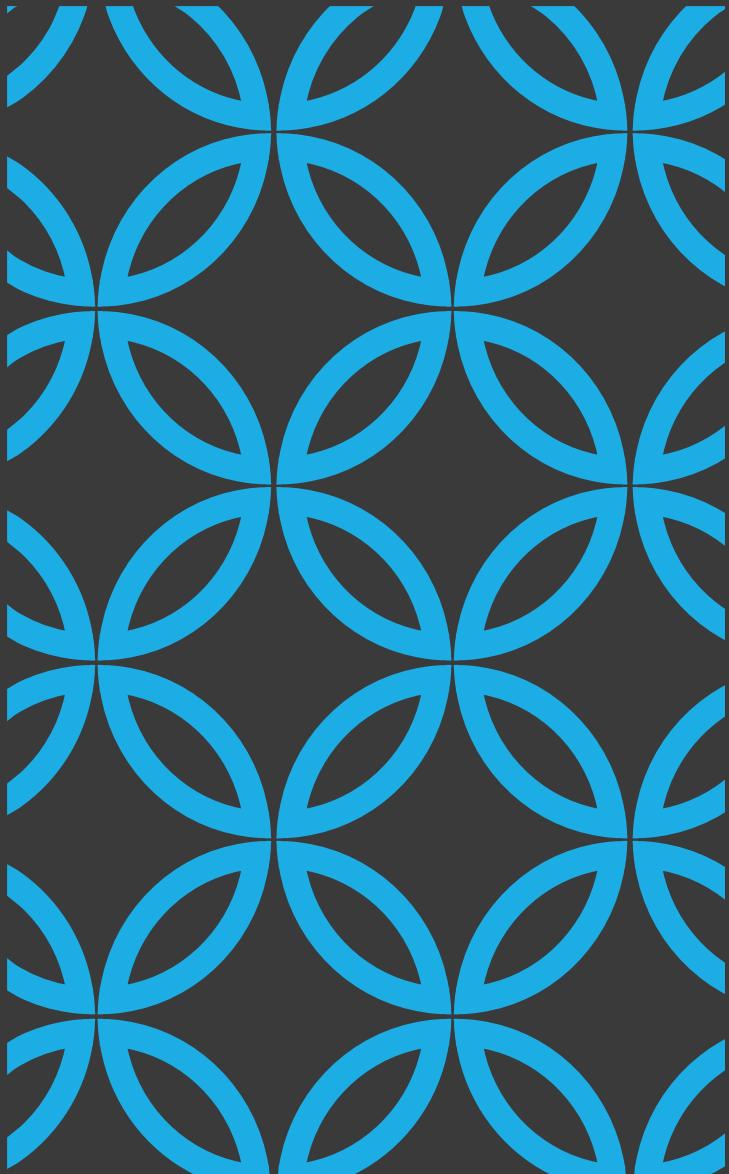
```
R  
I  
C  
O  
R  
A
```

String型も内部的にはリストっぽい構造になっている。

“RICORA” = [‘R’ , ‘I’ , ‘C’ , ‘O’ , ‘R’ , ‘A’]

ただし, PythonではString型は部分的に変更などはできない. (immutable)

“RICORA”[2] = 'X' とかはできない.



制御構文 IF

分岐処理の制御構文を学ぶ

if文

[記述方法]

```
if <条件文>:
```

　　■ <条件文>がTrueの場合に実行する文

```
else:
```

　　■ <条件文>がFalseの場合に実行する文

- <条件文>はBoolean型の値である。
- elseのブロックは省略が可能

if_example1.py

```
myOld = 18
```

```
if myOld >= 20:  
    print("成人")  
else:  
    print("未成年")
```

実行結果

```
未成年
```

if – elif 文

[記述方法]

if <条件文1> :

　　■ <条件文1> がTrueの場合に実行する文

elif <条件文2>:

　　■ <条件文2> がTrueの場合に実行する文 (<条件文1> はFalse)

else:

　　■ <条件文1>,<条件文2> が両方Falseの場合に実行する文

- elifのブロックは複数設置可能
- elseのブロックは省略が可能

if_greeting.py

```
time = 13
if (7 <= time) and (time < 12):
    print("Good Morning")
elif ( 12 <= time ) and ( time < 18 ):
    print("Good Afternoon")
elif ( 18 <= time ) and ( time < 24 ):
    print("Good Evening")
else:
    print("zzz")
```

実行結果

```
Good Afternoon
```

elif は else + if の略。

else + if で書くと、インデントが深くなって読みにくい。 elifを使おう。

elif <条件文>:
hogehoge



else:
if <条件文>:
hogehoge

練習問題2

https://ja.wikipedia.org/wiki/Fizz_Buzz

問題 Fizz_Buzz ゲームで正しくカウントするプログラムを作れ
[ゲームルール]

プレイヤーは円状に座る。最初のプレイヤーは「1」と数字を発言する。次のプレイヤーは直前のプレイヤーの次の数字(1大きい数字)を発言していく。

ただし、

- ◆ 3で割り切れる場合は「Fizz」
- ◆ 5で割り切れる場合は「Buzz」
- ◆ 15で割り切れる場合は「Fizz Buzz」

を数の代わりに発言しなければならない。

発言を間違えた者や、ためらった者は脱落となる。

[追加ルール] 50までやつたら終了

解答例

fizz_buzz.py

```
for i in range(1,51):
    if i % 15 == 0:
        print("FizzBuzz")
    elif i % 3 == 0:
        print("Fizz")
    elif i % 5 == 0:
        print("Buzz")
    else:
        print(i)
```

練習問題3

問題 素数を判定するプログラムを作れ

練習問題1同様、入力を受け取るプログラムとする。

プログラム冒頭で `n = int(input())` で入力を受け取り、
nが素数か判定しよう。

ヒント1 nを割り切る数がないかを調べる。

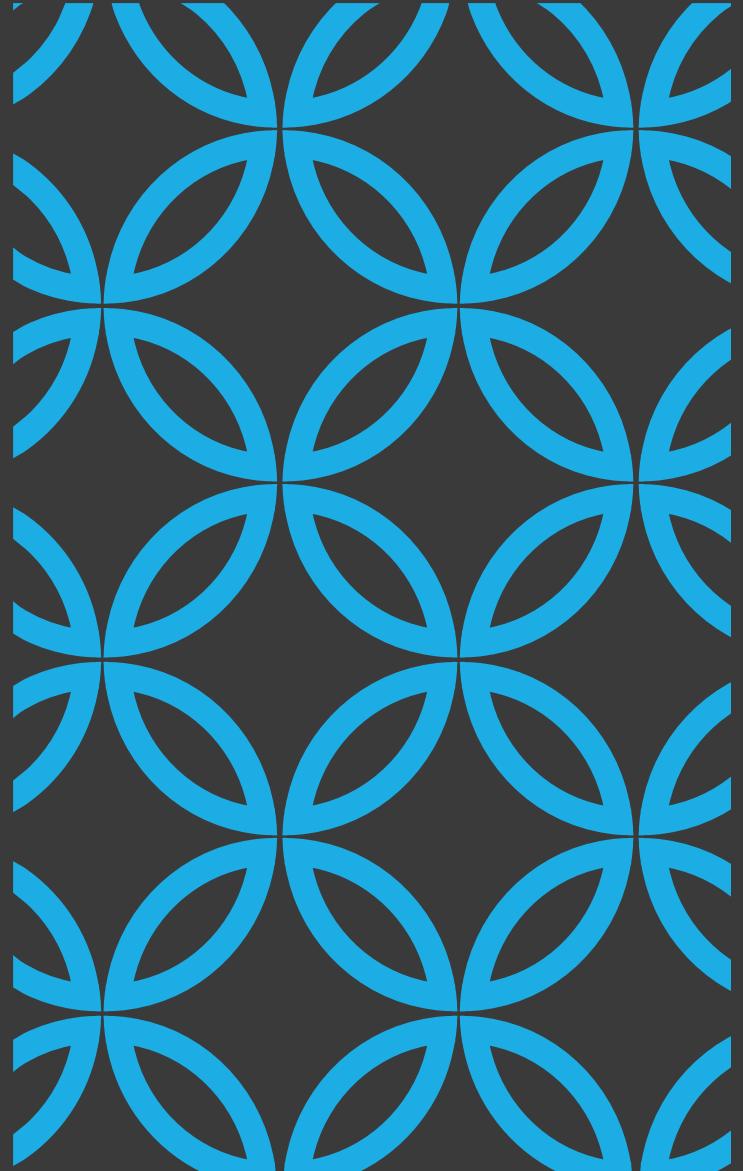
ヒント2 forループを使って総当たりする。

解答例

is_prime_number.py

```
n = int(input( ))  
is_prime = True
```

```
# nを割り切る数がないか調べる  
for i in range(2,n):  
    if n % i == 0:  
        is_prime = False  
  
if is_prime:  
    print("Prime Number")  
else:  
    print("Composite Number")
```



発展問題

問題 500以下の素数を全て出力せよ

ヒント 練習問題3を使う。

prime_number.py

```
for n in range(2, 500):
    is_prime = True

    # nを割り切る数がないか調べる
    for i in range(2,n):
        if n % i == 0:
            is_prime = False

    if is_prime:
        print(n)
```

解答例

最後に

この講座で扱えなかったけど、知っておきたい基礎知識

- コレクション
 - タプル (tuple), 辞書型 (dictionary)
- 制御構文
 - while文, ループ内でのbreakとcontinueの使い方
- その他
 - 関数 (定義の方法,呼び出しの方法), モジュールと標準ライブラリ, ファイル入出力

おすすめ参考書 & サイト (1)

■ Python入門本

「Python スタートブック」辻真吾(著)



■ 無料サイト

<https://www.python.ambitious-engineer.com/introduction-index>

■ Progate

オンラインプログラミング学習サービス(有料) <https://prog-8.com/>

■ Udemy

オンライン動画学習サイト(有料) <https://www.udemy.com/ja/>

おすすめ参考書 & サイト (2)

- AtCoder (サークルでも使う予定)
競技プログラミングのコンテストサイト
競プロの問題は、プログラミングの基本演習をするのにも良い。

<https://kenkoooo.com/atcoder/#/table//>

- AOJ (Aize Online Judge)
会津大学が作っているオンラインジャッジサイト

<https://onlinejudge.u-aizu.ac.jp/home>

- LeetCode
就職のコーディング試験対策 (?) サイト

<https://leetcode.com/>