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

9 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。

今日の豆知識: プログラミング言語でよく使うカッコたち

カッコ	通称	Python での主な使われ方
( )		**優先順位**, **関数**, タプル, etc.
[ ]		**リスト**, 辞書
{ }		辞書, 集合
< >		(ほぼ使わない)

英語では順に “parentheses”, “brackets”, “curly brackets”, “angle brackets” と言うことが多いようです。

今日の豆知識: プログラミング言語でよく使うカッコたち

カッコ	通称	Python での主な使われ方
( )	丸カッコ	**優先順位**, **関数**, タプル, etc.
[ ]		**リスト**, 辞書
{ }		辞書, 集合
< >		(ほぼ使わない)

英語では順に “parentheses”, “brackets”, “curly brackets”, “angle brackets” と言うことが多いようです。

今日の豆知識: プログラミング言語でよく使うカッコたち

カッコ	通称	Python での主な使われ方
( )	丸カッコ	**優先順位**, **関数**, タプル, etc.
[ ]	角カッコ	**リスト**, 辞書
{ }		辞書, 集合
< >		(ほぼ使わない)

英語では順に “parentheses”, “brackets”, “curly brackets”, “angle brackets” と言うことが多いようです。

今日の豆知識: プログラミング言語でよく使うカッコたち

カッコ	通称	Python での主な使われ方
( )	丸カッコ	**優先順位**, **関数**, タプル, etc.
[ ]	角カッコ	**リスト**, 辞書
{ }	波カッコ	辞書, 集合
< >		(ほぼ使わない)

英語では順に “parentheses”, “brackets”, “curly brackets”, “angle brackets” と言うことが多いようです。

今日の豆知識: プログラミング言語でよく使うカッコたち

カッコ	通称	Python での主な使われ方
( )	丸カッコ	**優先順位**, **関数**, タプル, etc.
[ ]	角カッコ	**リスト**, 辞書
{ }	波カッコ	辞書, 集合
< >	山カッコ	(ほぼ使わない)

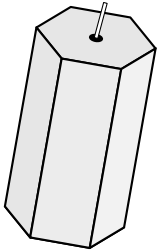
英語では順に “parentheses”, “brackets”, “curly brackets”, “angle brackets” と言うことが多いようです。

ウォーミングアップ

tiny おみくじ

問: プログラム先頭部分で定義される**msg0**, **msg1**, **msg2** の 3 変数が示す文字列のいずれかをランダムで出力する以下のプログラムを完成せよ。(3 分)

```
import random
msg0 = "daikichi"    # 大吉
msg1 = "kichi"       # 吉
msg2 = "kyo"         # 凶
# この続きを作る
```



解答例と次の課題

問:

解答例と次の課題

tiny おみくじ

```
import random
msg0 = "daikichi"
msg1 = "kichi"
msg2 = "kyo"
r = randint(0, 2)
if r == 0:
  print(msg0)
if r == 1:
  print(msg1)
if r == 2:
  print(msg2)
```

問:

解答例と次の課題

tiny おみくじ

```
import random
msg0 = "daikichi"
msg1 = "kichi"
msg2 = "kyo"
r = randint(0, 2)
if r == 0:
  print(msg0)
if r == 1:
  print(msg1)
if r == 2:
  print(msg2)
```

問: ここに中吉と小吉と半吉と末吉と末小吉と平(たいら)と小凶と半凶と末凶と大凶を加え、**deluxe おみくじ**とせよ。

解答例と次の課題

tiny おみくじ

```
import random
msg0 = "daikichi"
msg1 = "kichi"
msg2 = "kyo"
r = randint(0, 2)
if r == 0:
  print(msg0)
if r == 1:
  print(msg1)
if r == 2:
  print(msg2)
```

問: ここに中吉と小吉と半吉と末吉と末小吉と平(たいら)と小凶と半凶と末凶と大凶を加え、**deluxe おみくじ**とせよ。

いやそれ大変すぎ。

数学の問題っぽく書いたらこんなふう？

- $m_0 =$ 大吉, $m_1 =$ 吉, $m_2 =$ 中吉, ..., $m_{11} =$ 末凶, $m_{12} =$ 大凶 とする。

数学の問題っぽく書いたらこんなふう？

- $m_0 =$ 大吉, $m_1 =$ 吉, $m_2 =$ 中吉, ..., $m_{11} =$ 末凶, $m_{12} =$ 大凶 とする。
- 0 から 12 のいずれかの整数をランダムに選んだものを r とする。

数学の問題っぽく書いたらこんなふう？

- $m_0 =$ 大吉, $m_1 =$ 吉, $m_2 =$ 中吉, ..., $m_{11} =$ 末凶, $m_{12} =$ 大凶 とする。
- 0 から 12 のいずれかの整数をランダムに選んだものを r とする。
- このとき、**おみくじの値は m_r と表せる**。

数学の問題っぽく書いたらこんなふう？

- $m_0 =$ 大吉, $m_1 =$ 吉, $m_2 =$ 中吉, ..., $m_{11} =$ 末凶, $m_{12} =$ 大凶 とする。
- 0 から 12 のいずれかの整数をランダムに選んだものを r とする。
- このとき、**おみくじの値は m_r と表せる**。

山ほどの`if`がたったこれだけ!?

msgs
大吉
吉
中吉
小吉
リスト
[0]
[1]
[2]
[3]
[4]
[5]

リスト (aka 配列)

リストとは?

- 複数のデータを一列に並べて扱える便利なデータ
- 数値（インデックス）で要素にアクセス
- データの一種なのでもちろん変数に代入可

リストの使用例（とりあえず雰囲気だけ…）

```
# 月の英語表記を出力
months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
          "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]
m = int(input())    # 月を入力
print(months[m - 1])
```

リストその1: つくり方（リテラルとしての書き方）

角括弧で全体を囲んで要素をカンマで区切る（とっても簡単！）

```
[item0, item1, item2, item3,...]
```

地球型惑星リスト

```
planets = ["Mercury", "Venus", "Earth", "Mars"]
n_moons = [0, 0, 1, 2,]   #← 最後に, をつけてもいい
```

リストその1: つくり方（リテラルとしての書き方）

角括弧で全体を囲んで要素をカンマで区切る（とっても簡単！）

```
[item0, item1, item2, item3,...]
```

- 各要素の型は基本的に何でもいい。(数値、文字列はもちろん、リストも可。)

地球型惑星リスト

```
planets = ["Mercury", "Venus", "Earth", "Mars"]
n_moons = [0, 0, 1, 2,]   #← 最後に, をつけてもいい
```

リストその1: つくり方（リテラルとしての書き方）

角括弧で全体を囲んで要素をカンマで区切る（とっても簡単！）

```
[item0, item1, item2, item3,...]
```

- 各要素の型は基本的に何でもいい。(数値、文字列はもちろん、リストも可。)
- 要素が一つもない空リスト `[]` もアリ。

地球型惑星リスト

```
planets = ["Mercury", "Venus", "Earth", "Mars"]
n_moons = [0, 0, 1, 2,]   #← 最後に, をつけてもいい
```

リストその1: つくり方（リテラルとしての書き方）

角括弧で全体を囲んで要素をカンマで区切る（とっても簡単！）

```
[item0, item1, item2, item3,...]
```

- 各要素の型は基本的に何でもいい。(数値、文字列はもちろん、リストも可。)
- 要素が一つもない空リスト `[]` もアリ。
- 要素の型は全体で揃ってなくても良い。(例: `['ans', 42, []]`)

地球型惑星リスト

```
planets = ["Mercury", "Venus", "Earth", "Mars"]
n_moons = [0, 0, 1, 2,]   #← 最後に, をつけてもいい
```

リストその1: つくり方（リテラルとしての書き方）

角括弧で全体を囲んで要素をカンマで区切る（とっても簡単 !）

```
[item0, item1, item2, item3,...]
```

- 各要素の型は基本的に何でもいい。(数値、文字列はもちろん、リストも可。)
- 要素が一つもない空リスト `[]` もアリ。
- 要素の型は全体で揃ってなくても良い。(例: `['ans', 42, []]`)
- ふつうはその場で変数に代入する。(例: `depts = ['R', 'S', 'W']`)

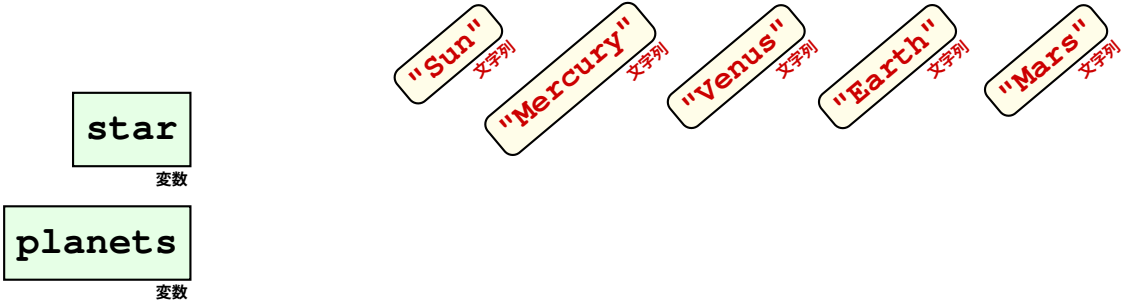
地球型惑星リスト

```
planets = ["Mercury", "Venus", "Earth", "Mars"]
n_moons = [0, 0, 1, 2,]   #← 最後に, をつけてもいい
```

リストのイメージ

「リスト = 複数の変数が一列に連なったカタマリ」というイメージで ok.

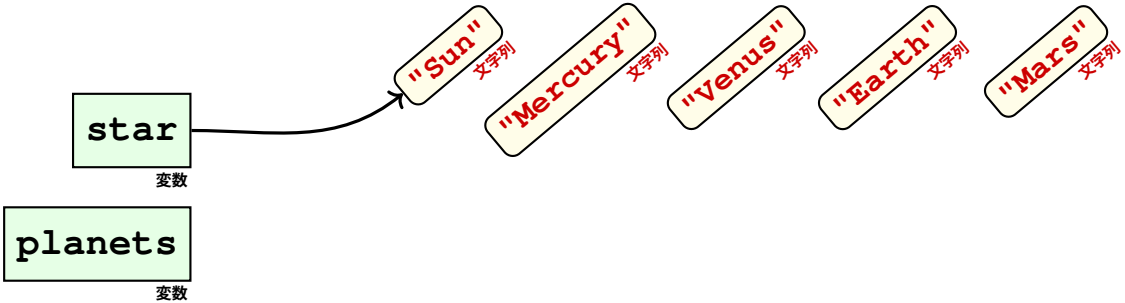
```python
# ssystem.py
star = "Sun"
planets = ["Mercury", "Venus", "Earth", "MarsU"]
print(star)
print(planets)
```



リストのイメージ

「リスト = 複数の変数が一列に連なったカタマリ」というイメージで ok.

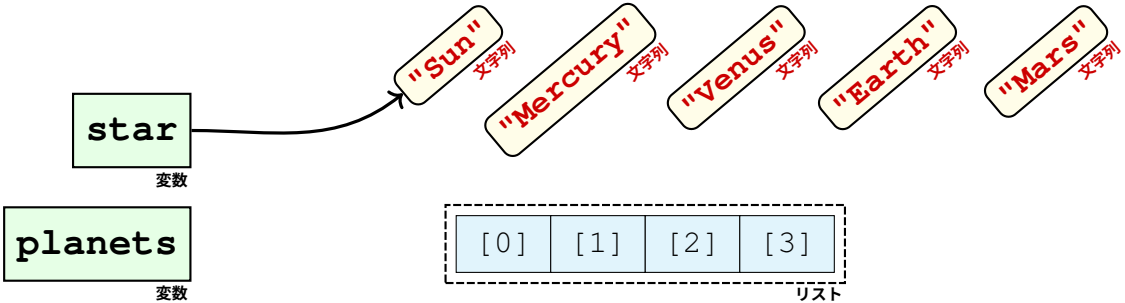
```python
# ssystem.py
star = "Sun"
planets = ["Mercury", "Venus", "Earth", "MarsU"]
print(star)
print(planets)
```



リストのイメージ

「リスト = 複数の変数が一列に連なったカタマリ」というイメージで ok.

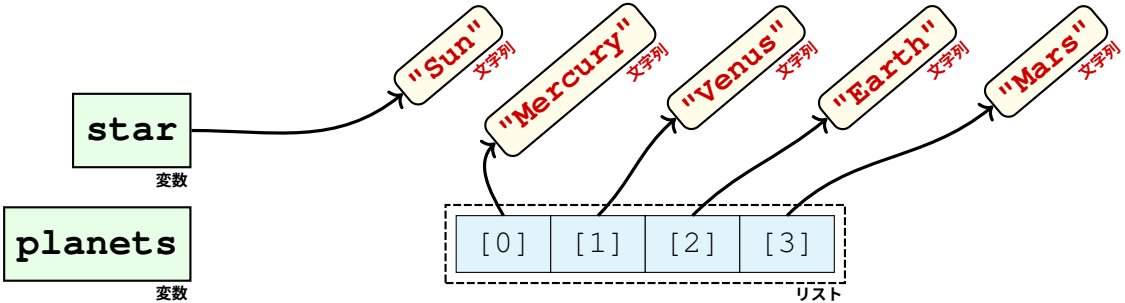
```python
# ssystem.py
star = "Sun"
planets = ["Mercury", "Venus", "Earth", "MarsU"]
print(star)
print(planets)
```



リストのイメージ

「リスト = 複数の変数が一列に連なったカタマリ」というイメージで ok.

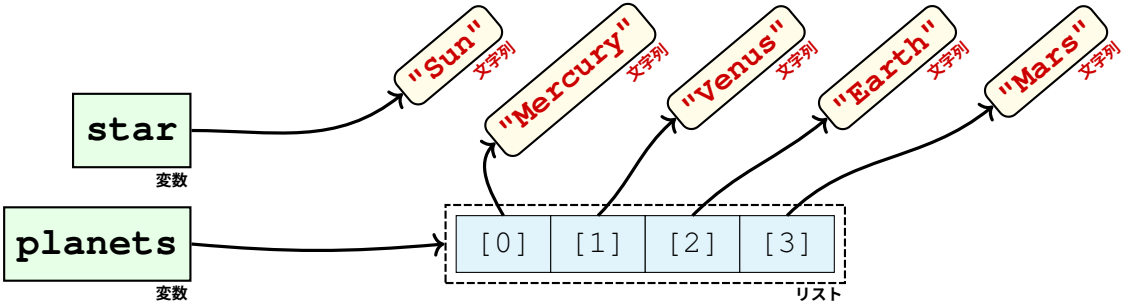
```python
# ssystem.py
star = "Sun"
planets = ["Mercury", "Venus", "Earth", "MarsU"]
print(star)
print(planets)
```



リストのイメージ

「リスト = 複数の変数が一列に連なったカタマリ」というイメージで ok.

```python
# ssystem.py
star = "Sun"
planets = ["Mercury", "Venus", "Earth", "MarsU"]
print(star)
print(planets)
```



リストその2: つかい方（インデックスでアクセス）

角括弧でインデックス指定（文字列といっしょ !）

```
planets[idx]
```

地球型惑星リスト

```
planets = ["Mercury", "Venus", "Earth", "Mars",]
n_moons = [0, 0, 1, 2,]
print(planets[3])
print(n_moons[-2])
print(len(planets))
print(planets[2][3])
```

リストその2: つかい方（インデックスでアクセス）

角括弧でインデックス指定（文字列といっしょ！）

planets[idx]

- 文字列のインデックス操作とほぼ同じ。

地球型惑星リスト

```
planets = ["Mercury", "Venus", "Earth", "Mars",]
n_moons = [0, 0, 1, 2,]
print(planets[3])
print(n_moons[-2])
print(len(planets))
print(planets[2][3])
```

リストその2: つかい方（インデックスでアクセス）

角括弧でインデックス指定（文字列といっしょ！）

planets[idx]

- 文字列のインデックス操作とほぼ同じ。
- 文字列と違って代入もできる。

地球型惑星リスト

```
planets = ["Mercury", "Venus", "Earth", "Mars",]
n_moons = [0, 0, 1, 2,]
print(planets[3])
print(n_moons[-2])
print(len(planets))
print(planets[2][3])
```

リストその2: つかい方（インデックスでアクセス）

角括弧でインデックス指定（文字列といっしょ！）

planets[idx]

- 文字列のインデックス操作とほぼ同じ。
- 文字列と違って代入もできる。
- **len()** も使える。

地球型惑星リスト

```
planets = ["Mercury", "Venus", "Earth", "Mars",]
n_moons = [0, 0, 1, 2,]
print(planets[3])
print(n_moons[-2])
print(len(planets))
print(planets[2][3])
```

リストその2: つかい方（インデックスでアクセス）

角括弧でインデックス指定（文字列といっしょ！）

planets[idx]

- 文字列のインデックス操作とほぼ同じ。
- 文字列と違って代入もできる。
- **len()** も使える。

地球型惑星リスト

```
planets = ["Mercury", "Venus", "Earth", "Mars",]
n_moons = [0, 0, 1, 2,]
print(planets[3])       #-> Mars
print(n_moons[-2])      #-> 1
print(len(planets))     #-> 4
print(planets[2][3])    #-> t
```

リストの練習

問. プログラムの続きを作り、実行例のように出力するようにせよ。(4 分)

```
planets = ["Mercury", "Venus", "Earth", "Mars",]
n_moons = [0, 0, 1, 2,]
```

実行例

```
Mercury has 0 moon(s).
Venus has 0 moon(s).
Earth has 1 moon(s).
Mars has 2 moon(s).
```

リストのありがたみを実感

これならいつ冥王星が復活しても大丈夫！

問. 以下のデータ (2025 年 5 月現在, wikipedia 調べ) を使って、木星型惑星についても対応せよ。(0〜2 分!)

惑星	木星 Jupiter	土星 Saturn	天王星 Uranus	海王星 Neptune
衛星数	97	274	28	16

リストは便利

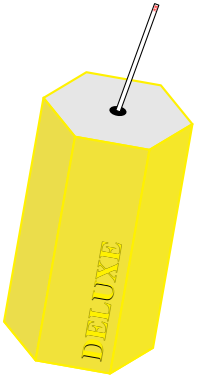
問. 各惑星の衛星数を出力したら、最後に衛星数の合計を出力するようにせよ。

実行例

```
Mercury has 0 moon(s).
Venus has 0 moon(s).
(中略)
Neptune has 14 moon(s).
There are 418 moons in the Solar System!
```

満を持して……
今なら楽勝♪

問. 大吉と 吉と 中吉と 小吉と 半吉と 末吉と 末小吉と 平と 凶と 小凶と 半凶と 末凶と 大凶からなる **deluxe おみくじ**を作れ。(5 分)



リストの演算ルール

リストも数値や文字列と同様に、データの一種なのでいくつかの演算ができます。

- リストと文字列は演算 ○
- リストと浮動小数点数は演算 ○
- リストと整数は演算 ○
- リストとリストは演算 ○

リストの演算ルール

リストも数値や文字列と同様に、データの一種なのでいくつかの演算ができます。

- リストと文字列は演算**できない**。
- リストと浮動小数点数は演算　　。
- リストと整数は演算　　。
- リストとリストは演算　　。

リストの演算ルール

リストも数値や文字列と同様に、データの一種なのでいくつかの演算ができます。

- リストと文字列は演算**できない**。
- リストと浮動小数点数は演算**できない**。
- リストと整数は演算　　。
- リストとリストは演算　　。

リストの演算ルール

リストも数値や文字列と同様に、データの一種なのでいくつかの演算ができます。

- リストと文字列は演算**できない**。
- リストと浮動小数点数は演算**できない**。
- リストと整数は演算**できる場合もある**。
- リストとリストは演算**できる場合もある**。

リストの演算表

+	整	リ
整	整	err
リ	err	リ

*	整	リ
整	整	リ
リ	リ	err

(注) リストは–と/の演算はできません。

リストの計算ドリル

次の計算をしなさい。

1. `[1, 2, 3] + ["a", "b"] =`
2. `[1, 2, 3] * 2 =`
3. `["a"] * 3 + 2 * [0] =`
4. `[2 * 3] + 2 * [3] =`

リストの計算ドリル

次の計算をしなさい。

1. `[1, 2, 3] + ["a", "b"] =` `[1, 2, 3, "a", "b"]`
2. `[1, 2, 3] * 2 =`
3. `["a"] * 3 + 2 * [0] =`
4. `[2 * 3] + 2 * [3] =`

リストの計算ドリル

次の計算をしなさい。

1. `[1, 2, 3] + ["a", "b"] =` `[1, 2, 3, "a", "b"]`
2. `[1, 2, 3] * 2 =` `[1, 2, 3, 1, 2, 3]`
3. `["a"] * 3 + 2 * [0] =`
4. `[2 * 3] + 2 * [3] =`

リストの計算ドリル

次の計算をしなさい。

1. `[1, 2, 3] + ["a", "b"] =` `[1, 2, 3, "a", "b"]`
2. `[1, 2, 3] * 2 =` `[1, 2, 3, 1, 2, 3]`
3. `["a"] * 3 + 2 * [0] =` `["a", "a", "a", 0, 0]`
4. `[2 * 3] + 2 * [3] =`

リストの計算ドリル

次の計算をしなさい。

1. `[1, 2, 3] + ["a", "b"] =` `[1, 2, 3, "a", "b"]`
2. `[1, 2, 3] * 2 =` `[1, 2, 3, 1, 2, 3]`
3. `["a"] * 3 + 2 * [0] =` `["a", "a", "a", 0, 0]`
4. `[2 * 3] + 2 * [3] =` `[6, 3, 3]`

リストの計算ドリル

次の計算をしなさい。

1. `[1, 2, 3] + ["a", "b"] =` `[1, 2, 3, "a", "b"]`
2. `[1, 2, 3] * 2 =` `[1, 2, 3, 1, 2, 3]`
3. `["a"] * 3 + 2 * [0] =` `["a", "a", "a", 0, 0]`
4. `[2 * 3] + 2 * [3] =` `[6, 3, 3]`

リストの要素への代入

- **初期化**

```
# 衛星の数
n_moons = [0, 0, 1, 2, 95, 149, 27, 14]  # index は 0〜7
```

- **既存の要素の上書き**

- **新しい要素の追加は……**

リストの要素への代入

- **初期化**

```
# 衛星の数
n_moons = [0, 0, 1, 2, 95, 149, 27, 14]  # index は 0〜7
```

- **既存の要素の上書き**

```
# 土星に新しい衛星が 2 つ発見された！ (フィクションです。)
n_moons[5] = n_moons[5] + 2    # これはいたってふつう
```

- **新しい要素の追加は……**

リストの要素への代入

- **初期化**

```
# 衛星の数
n_moons = [0, 0, 1, 2, 95, 149, 27, 14]  # index は 0〜7
```

- **既存の要素の上書き**

```
# 土星に新しい衛星が 2 つ発見された！ (フィクションです。)
n_moons[5] = n_moons[5] + 2    # これはいたってふつう
```

- **新しい要素の追加は……**

```
# 冥王星（衛星数 =5）が惑星に復活！
n_moons[8] = 5    # => 残念！ エラー
```

リストの要素への代入

- **初期化**

```
# 衛星の数
n_moons = [0, 0, 1, 2, 95, 149, 27, 14]  # index は 0〜7
```

- **既存の要素の上書き**

```
# 土星に新しい衛星が 2 つ発見された！ (フィクションです。)
n_moons[5] = n_moons[5] + 2    # これはいたってふつう
```

- **新しい要素の追加は……↓ 長さ以上の部分には代入不可！**(解決策は次回以降)

```
# 冥王星（衛星数 =5）が惑星に復活！
n_moons[8] = 5    # => 残念！ エラー
```

練習 (filostr.py)

問. 5 つの文字列を入力すると、順番を逆にして出力するプログラム **filostr.py**を **リストを上手に使って**作成せよ。

難易度: ★★☆☆☆

ヒント: とりあえず最初に 5 つ分文字列が入るリストを作ってしまいましょう。中身は何でも良いのですが、文字列を代入する予定なので **変数 = ["", "", "", "", ""]** とでもしておくのがわかりやすくて良いでしょう。ちなみにリストの演算を使ってもっとコンパクトに書く方法もありますが、気がつきましたか…?

実行例:

```
Mercury     <- 入力 1
Venus       <- 入力 2
Earth       <- 入力 3
Mars        <- 入力 4
Jupiter     <- 入力 5
Jupiter     <- 出力 1
Mars        <- 出力 2
Earth       <- 出力 3
Venus       <- 出力 4
Mercury     <- 出力 5
```

さらに練習 (filostr.py ver 2.0)

問. 前問を改変し、5 つの文字列を入力すると、順番を逆にし、かつ **右端を揃えて 回文出力**(?) するプログラムを作成せよ。

難易度: ★★★★☆

学習のねらい: リストと多重ループを組み合わせた難問です。一つ一つは単純でも組み合わせになるとできなくなる初学者は多くいます。この問題が良いトレーニングになるでしょう。

実行例:

```
Mercury     <- 入力 1
Venus       <- 入力 2
Earth       <- 入力 3
Mars        <- 入力 4
Jupiter     <- 入力 5
retipuJ     <- 出力 1
   sraM     <- 出力 2
  htraE     <- 出力 3
  suneV     <- 出力 4
yrucreM     <- 出力 5
```

裏課題 1: 以下の仕様のプログラムを作成せよ。

作ってみたけど課題にはちょっと難しすぎたのでボツにしました。

geaverage.py

- 10 個の数値を入力すると、
- 平均値と
- 平均値以上で平均値に一番近い数を出力する。

採点基準は

- 平均値が出てればまず 1 点、
- 加えて「平均値以上で〜数」が出てれば 1 点、
- 全体が 20 行以下であれば 1 点

難易度: ★★★★★

実行例

```
3.14 <-入力
1    <-入力
5    <-入力
9    <-入力
2    <-入力
6    <-入力
5    <-入力
-3   <-入力
-5   <-入力
8.97 <-入力
3.211-> 出力（平均値）
5.0  -> 出力
```

不採用

裏課題 2: 以下の仕様のプログラムを作成せよ。

作ってみたけど授業進行的に時間不足で説明途中の可能性があるのでこれもボツ。

div5.py

5 個の整数値を入力すると、最後の (5 番目に入力した) 数で割り切れるものをすべて、入力した順に、出力する。

難易度: ★★★☆☆

実行例

```
123   <-入力
456   <-入力
78    <-入力
910   <-入力
4     <-入力 (これで割り切れるか?)
456   -> 出力 (4 で割り切れる)
4     -> 出力 (4 で割り切れる)
```

不採用

練習問題解答

ssystem.py

```
planets = ["Mercury", "Venus", "Earth", "Mars",
           "Jupiter", "Saturn", "Uranus", "Neptune"]
n_moons = [0, 0, 1, 2, 95, 149, 28, 16]
i = 0    # loop counter
s = 0    # total number of moons
while i < len(planets):
    print(planets[i] + " has " + str(n_moons[i]) + " moon(s).")
    s = s + n_moons[i]
    i = i + 1
print("There are " + str(s) + " moons in the Solar System!")
```

dx_oracle.py

```
import random
msgs = ["daikichi", "kichi", "chukichi", "shokichi",
        "hankichi", "suiekichi", "suieshokichi",
        "taira", "kyo", "shokyo", "hankyo",
        "suiekyo", "daikyo"]
r = random.randint(0, len(msgs) - 1)
print(msgs[r])
```

filostr.py

```python
# filostr.py ver 1.0
strs = [""] * 5
i = 0
while i < 5:
    strs[i] = input()
    i = i + 1
i = 4
while i >= 0:
    print(strs[i])
    i = i - 1
```

```python
# filostr.py ver 2.0
strs = [""] * 5
i = 0
maxlen = 0    # 最大の文字列長
while i < 5:
    strs[i] = input()
    if len(strs[i]) > maxlen:
        maxlen = len(strs[i])
    i = i + 1
i = 4
while i >= 0:
    tgt = strs[i]   # 何度か使うので tgt に代入
    s = ""
    j = 0
    while j < len(tgt):
        s = s + tgt[len(tgt) - j - 1]
        j = j + 1
    print(" " * (maxlen - len(tgt)) + s)
    i = i - 1
```

geaverage.py

```
xs = [0] * 10
i = 0
s = 0
y = 0     # 平均値以上の最小値（仮）
while i < len(xs):
    xs[i] = float(input())
    s = s + xs[i]
    if xs[i] > y:
        y = xs[i]
    i = i + 1
ave = s / len(xs)
i = 0
while i < len(xs):
    if xs[i] >= ave and xs[i] < y:
        y = xs[i]
    i = i + 1
print(ave)
print(y)
```

div5.py

```
xs = [0] * 5
i = 0
while i < len(xs):
    xs[i] = int(input())
    i = i + 1
i = 0
while i < len(xs):
    if xs[i] % xs[-1] == 0:
        print(xs[i])
    i = i + 1
```