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。

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 **授業当日になるまで行わないように** してください。

プログラミング演習 I
Exercise on Programming I

本スライドは Teams で共有しています。

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

4 of 14

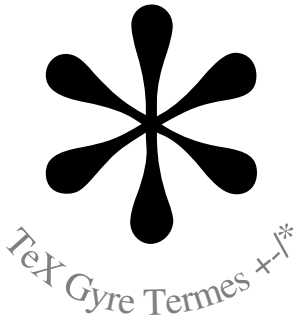
a LaTeX + Beamer slideshow

評価方法 update

『演習』の授業は、出席して、授業に参加すること自体がとても重要です。

- **演習出席点 =3 点/回** (30 分以内の遅刻 =2 点, 途中無断退出 =0 点。出席管理システムで採点するので必ず学生証を通すこと。学生証を忘れたら**授業中に SA もしくは教員に申し出ること。**)
- **レポート点 =3 点/回**
- **期末演習点 =16 点**
- **(new)** #10, 11, 12, 13, 14 は **演習欠席点 =−3 点/回** (特別な理由のない自己都合による欠席の場合)
  (なぜ？) 前半はとても易しいので内容的に本来は出席点として 3 点は多すぎます。一方で後半は 3 点では少なすぎます。かと言って後半に大きめの配点をしてしまうと全体の成績が悪くなってしまいます。そこでこのような減点で説明することにしました。ちょっと苦しいですが大多数の幸福のため。

フォントによるグリフの (大きな) 違い



どちらもASTERISK (U+002a) です。
（アスタリスク）

【#01 の復習】hello, world を詳しく見る。

```
print("hello, world")
```

- **print** は**何か**を表示する**関数**。
- **print (何か)** のように丸括弧を使う。
- **何か** の部分として、「引用符 (") で好きな言葉を囲んだもの (**文字列**) 」が書ける。
- **何か** のことを **引数**(ひきすう) と言う。

【#02 の復習】文字列と計算式と `print()`

- `print(文字列)`
- `print(計算式)`

文字列

引用符記号で囲んだ文字の並び
例: `"hello, world"`

計算式 (厳密な用語ではありません)

数字や括弧や演算子記号などの並びによる計算式
例: `(2 + 4) * 7`

print の深い闇

前回 (#03) のコード

```
print(10 + 2)
print()
print(10+2)
```

の 2 行目は説明があったとおり**改行（空行を出力）する**という**print**の使い方。

その説明を聞いて、

- **すんなり納得できた**貴方は
- 違和感・気持ち悪さ・不安を感じて**イマイチ納得できなかった**貴方は

print の深い闇

前回 (#03) のコード

```
print(10 + 2)
print()
print(10+2)
```

の 2 行目は説明があったとおり**改行（空行を出力）する**という`print`の使い方。

その説明を聞いて、

- **すんなり納得できた**貴方はまぁそれでいい。
- 違和感・気持ち悪さ・不安を感じて**イマイチ納得できなかった**貴方は

print の深い闇

前回 (#03) のコード

```
print(10 + 2)
print()
print(10+2)
```

の 2 行目は説明があったとおり**改行（空行を出力）する**という**print**の使い方。

その説明を聞いて、

- **すんなり納得できた**貴方はまぁそれでいい。
- 違和感・気持ち悪さ・不安を感じて**イマイチ納得できなかった**貴方は**プログラミングのセンスあり**! 自信を持とう。

Q. 何と答える?

「3」と「5」を**合わせる**となに?

すまん、聞き方が悪かった。

こういうのをいわゆる『プロンプトエンジニアリング』と言うのでしょうか？

すまん、聞き方が悪かった。

こういうのをいわゆる『プロンプトエンジニアリング』と言うのでしょうか？

- **数として**「3」と「5」を合わせるとなに?

すまん、聞き方が悪かった。

こういうのをいわゆる『プロンプトエンジニアリング』と言うのでしょうか？

- **数として**「3」と「5」を合わせるとなに?
- **文字として**「3」と「5」を合わせるとなに?

すまん、聞き方が悪かった。

こういうのをいわゆる『プロンプトエンジニアリング』と言うのでしょうか？

- **数として**「3」と「5」を合わせるとなに?
- **文字として**「3」と「5」を合わせるとなに?
- 「3」と「5」を**加算する**とな に?

すまん、聞き方が悪かった。

こういうのをいわゆる『プロンプトエンジニアリング』と言うのでしょうか？

- **数として**「3」と「5」を合わせるとなに?
- **文字として**「3」と「5」を合わせるとなに?
- 「3」と「5」を**加算する**となに?
- 「3」と「5」を**並べる**となに?

Python 言語における「データ」の種類 →

重要

データには、　　と　　がある。

- `"hello"`や`'oit'`や`'3'`や`""`のように引用符で囲んだデータは**文字列**。＿＿＿＿でも文字**列**。＿＿＿＿でも文字**列**。
- `42`や`-3.14`や`1.6e-19`のように、**ふつうに計算できそうなデータ**は**数値**。
- どちらも`print()`関数で出力 (表示) できる。
- **文字列と数値の本質的な違いは　　にある。**

重要

データには、文字列と　　がある。

- `"hello"`や`'oit'`や`'3'`や`""`のように引用符で囲んだデータは**文字列**。＿＿＿＿でも文字**列**。＿＿＿＿でも文字**列**。
- `42`や`-3.14`や`1.6e-19`のように、**ふつうに計算できそうなデータ**は**数値**。
- どちらも`print()`関数で出力 (表示) できる。
- **文字列と数値の本質的な違いは　　にある。**

重要

データには、文字列と数値がある。

- `"hello"`や`'oit'`や`'3'`や`""`のように引用符で囲んだデータは**文字列**。＿＿＿＿でも文字**列**。＿＿＿＿でも文字**列**。
- `42`や`-3.14`や`1.6e-19`のように、**ふつうに計算できそうなデータ**は**数値**。
- どちらも`print()`関数で出力 (表示) できる。
- **文字列と数値の本質的な違いは　　にある。**

重要

データには、文字列と数値がある。

- `"hello"`や`'oit'`や`'3'`や`""`のように引用符で囲んだデータは**文字列**。1 文字 でも文字**列**。＿＿＿＿ でも文字**列**。
- `42`や`-3.14`や`1.6e-19`のように、**ふつうに計算できそうなデータ**は**数値**。
- どちらも`print()`関数で出力 (表示) できる。
- **文字列と数値の本質的な違いは　　にある。**

重要

データには、文字列と数値がある。

- `"hello"`や`'oit'`や`'3'`や`""`のように引用符で囲んだデータは**文字列**。1 文字 でも文字**列**。0 文字 でも文字**列**。
- `42`や`-3.14`や`1.6e-19`のように、**ふつうに計算できそうなデータ**は**数値**。
- どちらも`print()`関数で出力 (表示) できる。
- **文字列と数値の本質的な違いは　　にある。**

重要

データには、文字列と数値がある。

- `"hello"`や`'oit'`や`'3'`や`""`のように引用符で囲んだデータは**文字列**。1 文字 でも文字**列**。0 文字 でも文字**列**。
- `42`や`-3.14`や`1.6e-19`のように、**ふつうに計算できそうなデータ**は**数値**。
- どちらも`print()`関数で出力 (表示) できる。
- **文字列と数値の本質的な違いは演算 (計算)ルールにある。**

Python 言語における「データ」の種類 → 『型 (type)』

重要

データには、文字列と数値がある。

- `"hello"`や`'oit'`や`'3'`や`""`のように引用符で囲んだデータは**文字列**。1 文字 でも文字**列**。0 文字 でも文字**列**。
- `42`や`-3.14`や`1.6e-19`のように、**ふつうに計算できそうなデータ**は**数値**。
- どちらも`print()`関数で出力 (表示) できる。
- **文字列と数値の本質的な違いは演算 (計算)ルールにある。**

string

文字列で計算 !?

'ABC' + 'xyz' = ?

いきなり、結論。

- **たし算 (+)**と**かけ算 (*)**は条件つきで**文字列の計算**ができる。
- たし算にできること:

  +

- かけ算にできること:

  *****

いきなり、結論。

- **たし算 (+)**と**かけ算 (*)**は条件つきで**文字列の計算**ができる。
- たし算にできること:

  文字列 + 文字列

- かけ算にできること:

  *****

いきなり、結論。

- **たし算 (+)**と**かけ算 (*)**は条件つきで**文字列の計算**ができる。
- たし算にできること:

  文字列 + 文字列
- かけ算にできること:

  文字列 * 整数値

やってみよう

calstr.py

```
print("OIT" + "_" + "R&D")
print("-_" * 42)
```

▶ 何が起こるか想像してから実行しよう。

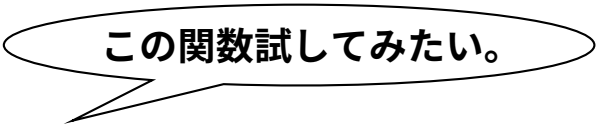
プログラムを書くまでもない、ちいさな『やってみよう』のお悩み

こ・れ・し・き・の・こ・と にいちいち

1. 新規ファイルを開き、
2. `print()` で結果を出力するプログラム書いて、
3. 端末で実行

なんてめんどくさいこと (cf. 前のページみたいなこと) やりたくない！

プログラムを書くまでもない、ちいさな『やってみよう』のお悩み

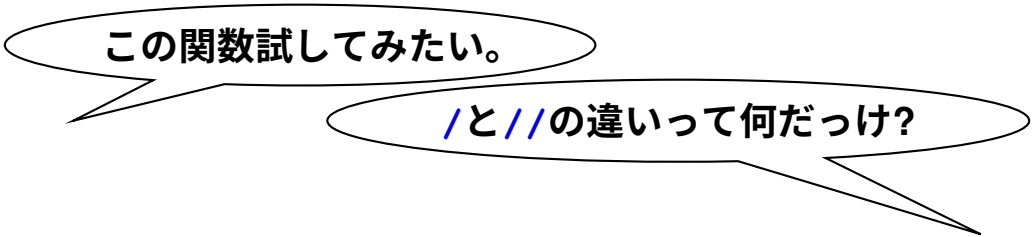


こ・れ・し・き・の・こ・と にいちいち

1. 新規ファイルを開き、
2. `print()`で結果を出力するプログラム書いて、
3. 端末で実行

なんてめんどくさいこと (cf. 前のページみたいなこと) やりたくない！

プログラムを書くまでもない、ちいさな『やってみよう』のお悩み



こ・れ・し・き・の・こ・と にいちいち

1. 新規ファイルを開き、
2. `print()`で結果を出力するプログラム書いて、
3. 端末で実行

なんてめんどくさいこと (cf. 前のページみたいなこと) やりたくない！

そんな貴方に『REPL』

Read-Eval-Print Loop 略して REPL

[a]厳密に一行というわけでもないのですが、今は一行という理解で ok。

実際の操作例は次ページ参照。

そんな貴方に『REPL』

Read-Eval-Print Loop 略して REPL

1. **ファイル名なしで python3 コマンド**を実行 → 対話モードに入る。

[a] 厳密に一行というわけでもないのですが、今は一行という理解で ok。

実際の操作例は次ページ参照。

そんな貴方に『REPL』

Read-Eval-Print Loop 略して REPL

1. **ファイル名なしで python3 コマンド**を実行 → 対話モードに入る。
2. 一行[a]入力する (つまりコンピュータが**読み込む**(r e a d)) と、

[a]厳密に一行というわけでもないのですが、今は一行という理解で ok。

実際の操作例は次ページ参照。

そんな貴方に『REPL』

Read-Eval-Print Loop 略して REPL

1. **ファイル名なしで python3 コマンド**を実行 → 対話モードに入る。
2. 一行[a]入力する (つまりコンピュータが**読み込む**(read)) と、
3. **その場で即、評価 (実行)**(evaluate)され、

[a]厳密に一行というわけでもないのですが、今は一行という理解で ok。

実際の操作例は次ページ参照。

そんな貴方に『REPL』

Read-Eval-Print Loop 略して REPL

1. **ファイル名なしで python3 コマンド**を実行 → 対話モードに入る。
2. 一行[a]入力する (つまりコンピュータが**読み込む**(read)) と、
3. **その場で即、評価 (実行)**(evaluate)され、
4. **その結果が出力**(print)され、

[a] 厳密に一行というわけでもないのですが、今は一行という理解で ok。

実際の操作例は次ページ参照。

そんな貴方に『REPL』

Read-Eval-Print Loop 略して REPL

1. **ファイル名なしで python3 コマンド**を実行 → 対話モードに入る。
2. 一行[a]入力する (つまりコンピュータが**読み込む**(read)) と、
3. **その場で即、評価 (実行)**(evaluate)され、
4. **その結果が出力**(print)され、
5. ふたたび**2. に戻る**(loop)。

[a] 厳密に一行というわけでもないのですが、今は一行という理解で ok。

実際の操作例は次ページ参照。

そんな貴方に『REPL』

Read-Eval-Print Loop 略して REPL

1. **ファイル名なしで python3 コマンド**を実行 → 対話モードに入る。
2. 一行[a]入力する (つまりコンピュータが**読み込む**(read)) と、
3. **その場で即、評価 (実行)**(evaluate)され、
4. **その結果が出力**(print)され、
5. ふたたび**2. に戻る**(loop)。
6. もういいや、これで終了、と思ったら **[Control] + [D]**で抜ける。

[a]厳密に一行というわけでもないのですが、今は一行という理解で ok。

実際の操作例は次ページ参照。

REPLの使用例

```
hkoba@mypc prog1 % python3
Python 3.12.3 (main, Apr  9 2024, 08:09:14) [Clang 15.0.0 (clang-15
Type "help", "copyright", "credits" or "license" for more informati
>>> "OIT" + "_" + "R&D"
'OIT_R&D'
>>> "-_" * 42
'-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
>>> ^D              ← [Control] + [D] を入力するとこう表示される
hkoba@mypc prog1 %
```

練習: REPLで厚さ0.1mmのコピー用紙を100回折り畳むとどのくらいの厚みになるか計算してみよう！

文字列の演算についての注意

要は 2 ページ前の 2 パタンのみ

- `"12345" - "45"`
- `"121212" / "12"`
- `42 * "hello"`
- `"OIT"*3 + "R&D"`

REPL で試してみよう！(そしてエラーメッセージを読んでみよう。)

文字列の演算についての注意

要は 2 ページ前の 2 パタンのみ

- `"12345" - "45"` のような引き算は無理。
- `"121212" / "12"`
- `42 * "hello"`
- `"OIT"*3 + "R&D"`

REPL で試してみよう！(そしてエラーメッセージを読んでみよう。)

文字列の演算についての注意

要は 2 ページ前の 2 パタンのみ

- `"12345" - "45"` のような引き算は無理。
- `"121212" / "12"` と、割り算も無理。
- `42 * "hello"`
- `"OIT"*3 + "R&D"`

REPL で試してみよう！(そしてエラーメッセージを読んでみよう。)

文字列の演算についての注意

要は 2 ページ前の 2 パタンのみ

- `"12345" - "45"` のような引き算は無理。
- `"121212" / "12"` と、割り算も無理。
- `42 * "hello"` は可。積は可換。(わかる?)
- `"OIT"*3 + "R&D"`

REPL で試してみよう！(そしてエラーメッセージを読んでみよう。)

文字列の演算についての注意

要は 2 ページ前の 2 パタンのみ

- `"12345" - "45"` のような引き算は無理。
- `"121212" / "12"` と、割り算も無理。
- `42 * "hello"` は可。積は可換。(わかる?)
- `"OIT"*3 + "R&D"` は問題なし。

REPL で試してみよう！(そしてエラーメッセージを読んでみよう。)

REPL じゃなくてふつうにプログラムを作る練習 (5 分)

ヒントは次ページにあるけれどできるだけ見ないで考えよう。

『`/`』で大四角形 (幅 40 文字縦 5 文字!) を書くプログラム **`mybox.py`** を作れ。(発展編: できた人は『`"`』で書いてみよう。)

実行結果

```
////////////////////////////////////////
/                                      /
/                                      /
/                                      /
////////////////////////////////////////
```

ヒント

1 行ずつ表示すれば高さ 5 の四角形が書ける。

書きたいものはこんなかたち:

1. 1 行目: 40 個の `/`
2. 2 行目: 1 個の `/` + 38 個のスペース + 1 個の `/`
3. 3 行目: 2 行目と同じ
4. 4 行目: 2 行目と同じ
5. 5 行目: 1 行目と同じ

mybox.py ver. 1.0

```
print("/" * 40)
print("/" + " " * 38 + "/")
print("/" + " " * 38 + "/")
print("/" + " " * 38 + "/")
print("/" * 40)
```

教訓

ここまでの話をもう少し一般化した話。いずれこの事実の重要さがわかるようになります。

データの型によって、演算の規則が変わる。

例: 同じ * でも、

- **数値**どうしは で、
- **文字列**と**数値**なら で、
- **文字列**どうしなら 。

教訓

ここまでの話をもう少し一般化した話。いずれこの事実の重要さがわかるようになります。

データの型によって、演算の規則が変わる。

例: 同じ * でも、

- **数値**どうしは**ふつうの乗算**で、
- **文字列**と**数値**なら　　で、
- **文字列**どうしなら　　。

教訓

ここまでの話をもう少し一般化した話。いずれこの事実の重要さがわかるようになります。

データの型によって、演算の規則が変わる。

例: 同じ * でも、

- **数値**どうしは**ふつうの乗算**で、
- **文字列**と**数値**なら**繰り返し**で、
- **文字列**どうしなら　　。

教訓

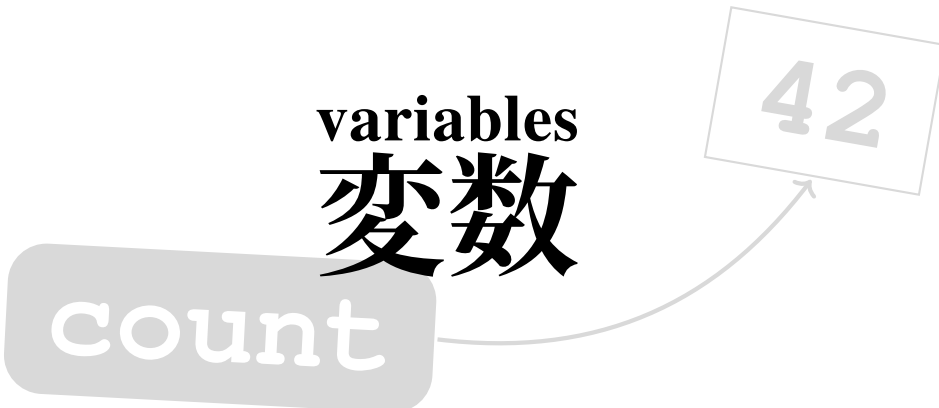
ここまでの話をもう少し一般化した話。いずれこの事実の重要さがわかるようになります。

データの型によって、演算の規則が変わる。

例: 同じ * でも、

- **数値**どうしは**ふつうの乗算**で、
- **文字列**と**数値**なら**繰り返し**で、
- **文字列**どうしなら**エラー**。

variables
変数



中１数学

順を追って答えよ。

順を追って答えよ。

1. $x = 6$ とする。(代入)

順を追って答えよ。

1. $x = 6$ とする。(代入)
2. $x \times 2$ を答えよ。

順を追って答えよ。

1. $x = 6$ とする。(代入)
2. $x \times 2$ を答えよ。
3. $y = 7$ とする。(代入)

順を追って答えよ。

1. $x = 6$ とする。(代入)
2. $x \times 2$ を答えよ。
3. $y = 7$ とする。(代入)
4. xy を答えよ。

順を追って答えよ。

1. $x = 6$ とする。
2. $x \times 2$ を答えよ。
3. $y = 7$ とする。(代入)
4. xy を答えよ。

Pythonでやろう。

Python で中１数学

1. $x = 6$ とする。(代入)
2. $x \times 2$ を答えよ。
3. $y = 7$ とする。(代入)
4. xy を答えよ。

Python で中１数学

1. $x = 6$ とする。(代入)
2. $x \times 2$ を答えよ。
3. $y = 7$ とする。(代入)
4. xy を答えよ。

calvar.py

```
x = 6
print(x * 2)
y = 7
print(x * y)
```

変数と言っても「数」だけじゃない件

こんなこともできます。

```
x = "OIT"
```

練習: 中１数学の１行目で、上記「こんなこと」をやってみよう。

変数で働き方改革

- あなたは『横 40 文字、縦 5 文字の四角形』職人です。

変数で働き方改革

- あなたは『横 40 文字、縦 5 文字の四角形』職人です。
- 先日『/』で仕事の依頼がありました。プログラムは完成しています。

変数で働き方改革

- あなたは『横 40 文字、縦 5 文字の四角形』職人です。
- 先日『**/**』で仕事の依頼がありました。プログラムは完成しています。
- 今日『**#**』で依頼がありました。これから作ります。

変数で働き方改革

- あなたは『横 40 文字、縦 5 文字の四角形』職人です。
- 先日『**/**』で仕事の依頼がありました。プログラムは完成しています。
- 今日『**#**』で依頼がありました。これから作ります。
- 明日『**x**』で依頼があるかも知れません。

変数で働き方改革

- あなたは『横 40 文字、縦 5 文字の四角形』職人です。
- 先日『**/**』で仕事の依頼がありました。プログラムは完成しています。
- 今日『**#**』で依頼がありました。これから作ります。
- 明日『**x**』で依頼があるかも知れません。

さて、どうする？

もちろん、こうする。〜mybox.py ver. 2.0 〜

さっき作った四角を書くプログラムを改変します。

もちろん、こうする。〜mybox.py ver. 2.0 〜

さっき作った四角を書くプログラムを改変します。

```
c = "#"     # 変数名は好きなもので ok
print(c * 40)
print(c + " "*38 + c)
print(c + " "*38 + c)
print(c + " "*38 + c)
print(c * 40)
```

続・変数で働き方改革

- あなたは『模様が自在な横 40 文字、縦 5 文字の四角形』職人です。

続・変数で働き方改革

- あなたは『模様が自在な横 40 文字、縦 5 文字の四角形』職人です。
- 先日『**横 40 文字**』で仕事の依頼がありました。プログラムは完成しています。

続・変数で働き方改革

- あなたは『模様が自在な横 40 文字、縦 5 文字の四角形』職人です。
- 先日『**横 40 文字**』で仕事の依頼がありました。プログラムは完成しています。
- 今日『**横 42 文字**』で依頼がありました。これから作ります。

続・変数で働き方改革

- あなたは『模様が自在な横 40 文字、縦 5 文字の四角形』職人です。
- 先日『**横 40 文字**』で仕事の依頼がありました。プログラムは完成しています。
- 今日『**横 42 文字**』で依頼がありました。これから作ります。
- 明日『**横 20 文字**』で依頼があるかも知れません。

続・変数で働き方改革

- あなたは『模様が自在な横 40 文字、縦 5 文字の四角形』職人です。
- 先日『**横 40 文字**』で仕事の依頼がありました。プログラムは完成しています。
- 今日『**横 42 文字**』で依頼がありました。これから作ります。
- 明日『**横 20 文字**』で依頼があるかも知れません。

さて、どうする？

当然、こうなる。〜mybox.py ver. 3.0 〜

さっき作った四角を書くプログラムをさらに改変します。

当然、こうなる。〜mybox.py ver. 3.0 〜

さっき作った四角を書くプログラムをさらに改変します。

```
c = "#"  # 変数名は好きなもので ok
w = 40   # もっとわかりやすい変数名でも良い
print(c * w)
print(c + " " * (w - 2) + c)
print(c + " " * (w - 2) + c)
print(c + " " * (w - 2) + c)
print(c * w)
```

Epilogue

- あなたは『模様と横幅が自在な縦 5 文字の四角形』職人です。

Epilogue

- あなたは『模様と横幅が自在な縦 5 文字の四角形』職人です。
- 先日『**縦 5 文字**』で仕事の依頼がありました。プログラムは完成しています。

Epilogue

- あなたは『模様と横幅が自在な縦 5 文字の四角形』職人です。
- 先日『**縦 5 文字**』で仕事の依頼がありました。プログラムは完成しています。
- 今日『**縦 8 文字**』で依頼がありました。これから作ります。

Epilogue

- あなたは『模様と横幅が自在な縦 5 文字の四角形』職人です。
- 先日『**縦 5 文字**』で仕事の依頼がありました。プログラムは完成しています。
- 今日『**縦 8 文字**』で依頼がありました。これから作ります。……あれ?

Epilogue

- あなたは『模様と横幅が自在な縦 5 文字の四角形』職人です。
- 先日『**縦 5 文字**』で仕事の依頼がありました。プログラムは完成しています。
- 今日『**縦 8 文字**』で依頼がありました。これから作ります。……あれ?

さぁ、大ピンチ！

Epilogue

- あなたは『模様と横幅が自在な縦 5 文字の四角形』職人です。
- 先日『**縦 5 文字**』で仕事の依頼がありました。プログラムは完成しています。
- 今日『**縦 8 文字**』で依頼がありました。これから作ります。……あれ?

さぁ、大ピンチ！

(少し後の授業 (`while`や`for`) に続く)

変数についての込み入った話

本当はもっといろいろあるけれどさしあたりこのくらいを押さえておけば Python 生活 3 年くらいはいける。

- 変数名 (`x`とか`y`とか) は**だいたい自由**に決めていい。何文字でもいい。(例: `jugemjugemgokonosurikire = 100`)
- 変数名は**先頭がアルファベットか記号、2 文字目以降は数字も可**がキホンと思っておけばいい。(例: `cz800c = 155000`)
- 記号は**アンダースコアのみ**使える。(例: `x-1`→NG, `x_1`→OK)
- アルファベットは小文字を使うのがふつう。(大文字も使えるが)
- 複数語からなる長い変数名は単語をアンダースコアで区切るのが慣習。(例: `enemy_life = 255`)

本当のもっといろいろを知りたい人は `https://docs.python.org/ja/3/reference/lexical_analysis.html` あたりを参照。

Python 変数名クイズ

Q. ○× を答えよ。

変数名	文法的に	慣習的に
`high_score`		
`highScore`		
`HighScore`		
`_score`		
`$score`		
`@score`		
`_`		

Python 変数名クイズ

Q. ○× を答えよ。

変数名	文法的に	慣習的に
`high_score`	○	○
`highScore`		
`HighScore`		
`_score`		
`$score`		
`@score`		
`_`		

Python 変数名クイズ

Q. ○× を答えよ。

変数名	文法的に	慣習的に
`high_score`	○	○
`highScore`	○	○
`HighScore`		
`_score`		
`$score`		
`@score`		
`_`		

Python 変数名クイズ

Q. ○× を答えよ。

変数名	文法的に	慣習的に
high_score	○	○
highScore	○	○
HighScore	○	(×)
_score		
$score		
@score		
_		

Python 変数名クイズ

Q. ○× を答えよ。

変数名	文法的に	慣習的に
`high_score`	○	○
`highScore`	○	○
`HighScore`	○	(×)
`_score`	○	(○)
`$score`		
`@score`		
`_`		

Python 変数名クイズ

Q. ○× を答えよ。

変数名	文法的に	慣習的に
high_score	○	○
highScore	○	○
HighScore	○	(×)
_score	○	(○)
$score	×	×
@score		
_		

Python 変数名クイズ

Q. ○× を答えよ。

変数名	文法的に	慣習的に
`high_score`	○	○
`highScore`	○	○
`HighScore`	○	(×)
`_score`	○	(○)
`$score`	×	×
`@score`	×	×
`_`		

Python 変数名クイズ

Q. ○× を答えよ。

変数名	文法的に	慣習的に
`high_score`	○	○
`highScore`	○	○
`HighScore`	○	(×)
`_score`	○	(○)
`$score`	×	×
`@score`	×	×
`_`	○	(○)

コラム: 変数 (variable) と定数 (constant)

雑談なので意味がわからなければ聞き／読み流してください。

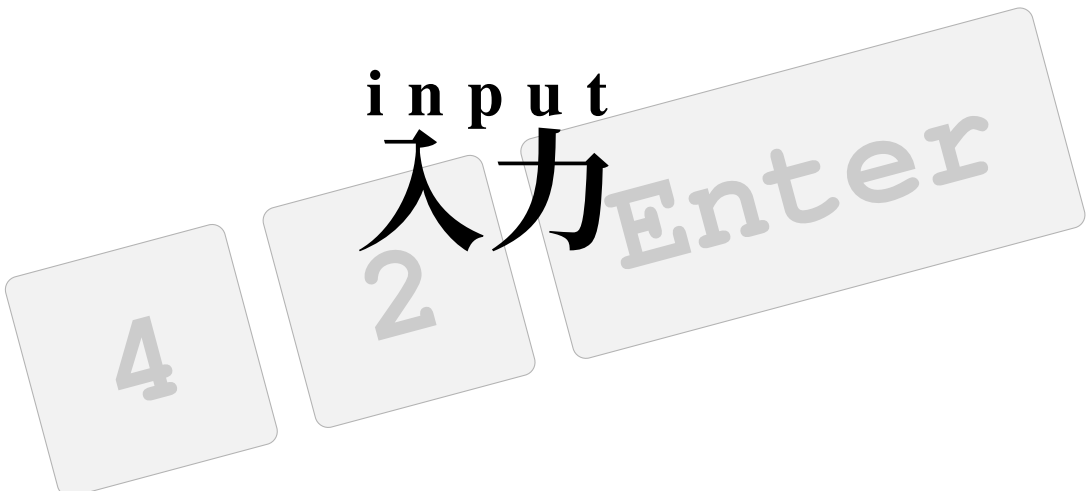
「働き方改革」の例題で挙げたコードには`c`や`w`という**変数**を用いました。あれ？でも変数って「**変わる**」数なのに一度も変わってないじゃん……、と思った貴方は鋭い。

そう、実はこの例では変数とは名ばかりで変わっていません。実は、このように『**実行中に一度も変わらない「変数的なもの」は「定数」として区別すべき**』という考え方があります。その方がプログラマの意図が明確になりバグが減り、機械的な最適化もやりやすく性能が向上するからです。

実際、多くの言語では**変数**とは別に**定数**という仕組みも導入しています。（その辺は知識として知っておくと良いでしょう。）ただ、残念ながら **Python に言語仕様としての定数はありません**。全部変数扱いです。

input

入力



文字列の入力

変数への決まった文字列の代入

```
S = "OIT"
```

`OIT`決め打ちじゃなくて、**キーボードから入力**したい！

変数への標準入力からの代入

```
S =
```

文字列の入力

変数への決まった文字列の代入

```
s = "OIT"
```

OIT決め打ちじゃなくて、**キーボードから入力**したい！

変数への標準入力からの代入

```
s = input()
```

input() の使用例

プログラム “sayhello.py”↓

```
n = input()                          # 名前を入力すると
print("Hello, " + n + "!")  # 挨拶してくれる！
```

実行例 ↓

```
hkoba@host:~/prog1$ python3 sayhello.py
kobayashi                # ← キーボードからの入力
Hello, kobayashi!        # ←print された出力
```

ほんのちょっと難しい練習

問. 標準入力（キーボード）から文字列 s を入力すると、**s を 5 個アンダースコア (_) でつなげて出力**するプログラム **replicate.py** を作成せよ。

```
hkoba@host:~/prog1$ python3 replicate.py
OIT                 # ← キーボードからの入力
OIT_OIT_OIT_OIT_OIT # ←print された出力
```

ほんのちょっと難しい練習

問. 標準入力（キーボード）から文字列 s を入力すると、**s を 5 個アンダースコア (_) でつなげて出力**するプログラム **replicate.py** を作成せよ。

```
hkoba@host:~/prog1$ python3 replicate.py
OIT                 # ← キーボードからの入力
OIT_OIT_OIT_OIT_OIT # ←print された出力
```

解答例:

```
s = input()
print((s + "_") * 4 + s)
```

要は `int()` を適用すればいい。

決まった文字列の代入

```
s = "42"
```

決まった数値の代入

```
x = 42
```

標準入力からの文字列代入

```
s =
```

標準入力からの (整) 数値代入

```
x =
```

※ `int()` を使うと**整数**のみが入力できます。浮動小数点数を入力するには `float()` を使うのですが、プログラミングのキホンはやっぱり整数なのでこの授業では整数をメインに扱います。

要は `int()` を適用すればいい。

決まった文字列の代入

```
s = "42"
```

決まった数値の代入

```
x = 42
```

標準入力からの文字列代入

```
s = input()
```

標準入力からの (整) 数値代入

```
x =
```

※ `int()` を使うと**整数**のみが入力できます。浮動小数点数を入力するには `float()` を使うのですが、プログラミングのキホンはやっぱり整数なのでこの授業では整数をメインに扱います。

要は `int()` を適用すればいい。

決まった文字列の代入

```
s = "42"
```

決まった数値の代入

```
x = 42
```

標準入力からの文字列代入

```
s = input()
```

標準入力からの (整) 数値代入

```
x = int(input())
```

※ `int()` を使うと**整数**のみが入力できます。浮動小数点数を入力するには `float()` を使うのですが、プログラミングのキホンはやっぱり整数なのでこの授業では整数をメインに扱います。

練習 (さっきの「ほんのちょっと難しい練習」ができていれば簡単)

問. **replicate.py**を改変して、標準入力（キーボード）から入力した文字列 s と数値 n を入力すると、**s を n 個**アンダースコア (_) でつなげて出力するようにせよ。

```
hkoba@host:~/prog1$ python3 replicate.py
OIT                         # ← キーボードからの入力 s
6                           # ← キーボードからの入力 n
OIT_OIT_OIT_OIT_OIT_OIT # ←print された出力
```

この辺まで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】

- 【標準入力サンプル問題セット】
  - 1 つのデータ入力
  - 1 行のデータ入力
  - 3 行のデータ入力
- 【D ランクレベルアップメニュー】
  - 2 つの数値を出力 (全問)
  - 代入演算 1(STEP 1, STEP 3, (FINAL))
    (STEP 2 は微妙に授業でやってない内容を含む。FINAL もそういう意味でちょっと微妙。)
  - 乗客人数 (STEP 1)


処理 P
if-statements
条件分岐
if cond?
False
True
処理 F
処理 T

`if` 降臨

当然かつ重要なこととして言ってきた、

『プログラムは、　　、実行される』

の基本**ルールを、今から打ち破る**。それが`if`。

`if` 降臨

当然かつ重要なこととして言ってきた、

『プログラムは、**上から順番に**、実行される』

の基本**ルールを、今から打ち破る**。それが`if`。

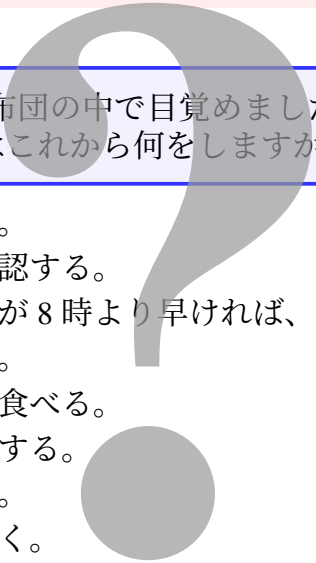
日本語文章の読解問題

現在 8:10 分で、あなたは布団の中で目覚めました。すべきこと(プログラム)は以下のとおりです。さて、あなたはこれから何をしますか?

1. 目覚める。
2. 時刻を確認する。
3. **もし**時刻が 8 時より早ければ、
4. 顔を洗う。
5. 朝ご飯を食べる。
6. 歯磨きをする。
7. 着替える。
8. 大学に行く。

日本語文章の読解問題

現在 8:10 分で、あなたは布団の中で目覚めました。すべきこと(プログラム)は以下のとおりです。さて、あなたはこれから何をしますか?



1. 目覚める。
2. 時刻を確認する。
3. **もし**時刻が 8 時より早ければ、
4. 顔を洗う。
5. 朝ご飯を食べる。
6. 歯磨きをする。
7. 着替える。
8. 大学に行く。

曖昧な`if`解消への挑戦その 1 〜キーワードで分ける〜

1. 目覚める。
2. 時刻を確認する。
3. **もし**時刻が 8 時より早ければ、
4. 顔を洗う。
5. 朝ご飯を食べる。
6. 歯磨きをする。
7. 着替える。
8. 大学に行く。

曖昧な`if`解消への挑戦その 1 〜キーワードで分ける〜

1. 目覚める。
2. 時刻を確認する。
3. **もし**時刻が 8 時より早ければ、
4. 顔を洗う。
5. 朝ご飯を食べる。
6. 歯磨きをする。**end**
7. 着替える。
8. 大学に行く。

曖昧な`if`解消への挑戦その 1 〜キーワードで分ける〜

1. 目覚める。
2. 時刻を確認する。
3. **もし**時刻が 8 時より早ければ、
4. 顔を洗う。
5. 朝ご飯を食べる。
6. 歯磨きをする。**end**
7. 着替える。
8. 大学に行く。

```ruby
# 本当に動く Ruby プログラム
puts('目覚めました')
t = Time.now
if t.hour < 8
puts('顔を洗いました')
puts('朝ご飯を食べました')
puts('歯磨きしました')
end
puts('着替えました')
puts('出発しました')
```

注: ふつうは`if`ブロックの中は字下げするものですが、ここではキーワードで分けるという意味を強調するために敢えて字下げなしで書いてあります。

曖昧な`if`解消への挑戦その 2 〜記号で分ける〜

1. 目覚める。
2. 時刻を確認する。
3. **もし**時刻が 8 時より早ければ
4. 顔を洗う。
5. 朝ご飯を食べる。
6. 歯磨きをする。
7. 着替える。
8. 大学に行く。

曖昧な`if`解消への挑戦その 2 〜記号で分ける〜

1. 目覚める。
2. 時刻を確認する。
3. **もし**時刻が 8 時より早ければ『
4. 顔を洗う。
5. 朝ご飯を食べる。
6. 歯磨きをする。』
7. 着替える。
8. 大学に行く。

曖昧なif解消への挑戦その 2 〜記号で分ける〜

1. 目覚める。
2. 時刻を確認する。
3. もし時刻が 8 時より早ければ『
4. 顔を洗う。
5. 朝ご飯を食べる。
6. 歯磨きをする。』
7. 着替える。
8. 大学に行く。

```
// 本当に動く JavaScript プログラム
console.log('目覚めました');
t = new Date();
if (t.getHours() < 8) {
console.log('顔を洗いました');
console.log('朝ご飯を食べました');
console.log('歯磨きしました');
}
console.log('着替えました');
console.log('出発しました');
```

注: ふつうはifブロックの中は字下げするものですが、ここではキーワードで分けるという意味を強調するために敢えて字下げなしで書いてあります。

そして、Python は……

曖昧な`if`解消への挑戦その 3 ～字下げ（インデント）で分ける～

1. 目覚める。
2. 時刻を確認する。
3. **もし**時刻が 8 時より早ければ、
4. 顔を洗う。
5. 朝ご飯を食べる。
6. 歯磨きをする。
7. 着替える。
8. 大学に行く。

曖昧なif解消への挑戦その 3 ～字下げ(インデント)で分ける～

1. 目覚める。
2. 時刻を確認する。
3. もし時刻が 8 時より早ければ、
4. 　　顔を洗う。
5. 　　朝ご飯を食べる。
6. 　　歯磨きをする。
7. 着替える。
8. 大学に行く。

曖昧なif解消への挑戦その 3 〜字下げ(インデント)で分ける〜

1. 目覚める。
2. 時刻を確認する。
3. **もし**時刻が 8 時より早ければ、
4. 　　顔を洗う。
5. 　　朝ご飯を食べる。
6. 　　歯磨きをする。
7. 着替える。
8. 大学に行く。

```
# 本当に動く Python プログラム
import time
print('目覚めました')
t = time.localtime()
if t.tm_hour < 8:
    print('顔を洗いました')
    print('朝ご飯を食べました')
    print('歯磨きしました')
print('着替えました')
print('出発しました')
```

Python の `if` の書き方

```
if                    :
```

Python の `if` の書き方

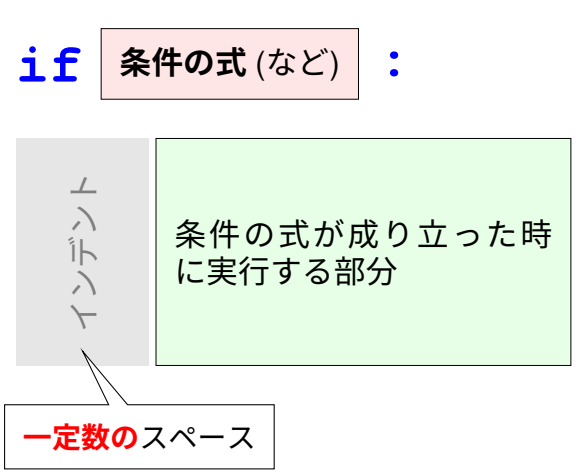
`if` **条件の式** (など) `:`

Python の `if` の書き方

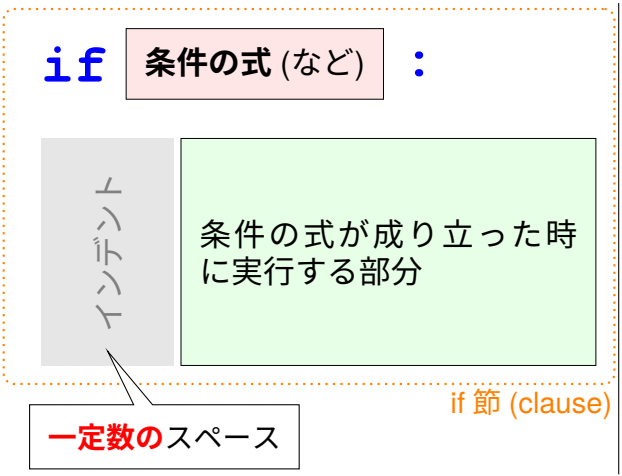
`if` **条件の式** (など) `:`

条件の式が成り立った時に実行する部分

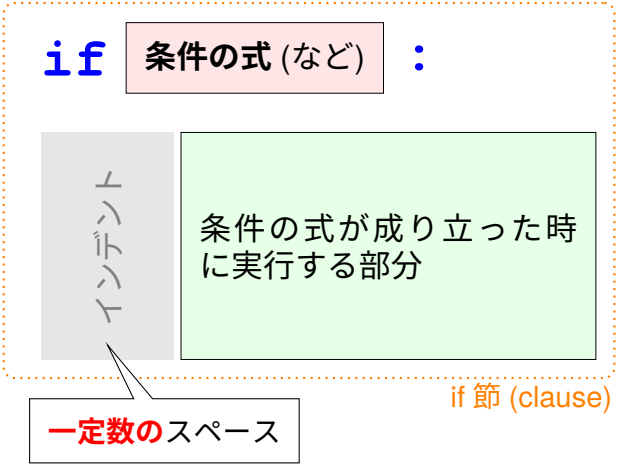
Python の `if` の書き方



Python の `if` の書き方



Python の `if` の書き方



簡単だから頑張って読んでみよう ↓。

```
print("trick or treat?")
ans = input()
if ans == "trick":
  print("You've got tricked!!")
  print("GAME OVER")
if ans == "treat":
  print("You earned candies!")
  print("GJ!")
print("The end")
```

「条件の式」の基本の書き方

(注) Python には「条件式」というこれとは別の特殊な式があるのですが、それとは別。

1. a と b が**等しいか?** ..... `a == b`
2. a と b が**等しくないか?** ..... `a != b`
3. a が b より**大きいか?** ..... `a > b`
4. a が b より**小さいか?** ..... `a < b`
5. a が b **以上か?** ..... `a >= b`
6. a が b **以下か?** ..... `a <= b`

概ね常識的な書き方で ok！

「条件の式」の書き方について補足

ポイント

関係演算子 (AKA 比較演算子) は、 `>`, `<`, `==`, `>=`, `<=`, `!=` の**6 つだけ**。

問題ない書き方

```
x == 3
42 == ans
z >= 0
a < 3
c != b
```

(可能だが)激しく非推奨

```
3 < x < 9
0 <= t <= 1
x == y != z
0 > u <= v
a < b <= c > 0
```

よくある間違い

```
a = 123
s =! 7
y =< 2020
8 => hour
a <> b
```

(参考) $0 \leq t \leq 1$ みたいな条件は次回やる予定の`and`, `or`を使うことを推奨します。

コラム: モダンな開発環境での関係演算子などの表示

数学では $a \leqq b$ と書くところを、ほとんどのプログラミング言語では `a <= b` と書きます。変ですね。なぜでしょう? 実は理由は単純で昔のコンピュータには $\leqq$ という**文字がなかった**、とそれだけです。今のコンピュータにはもちろん $\leqq$ はありますが、少なくともキーボードから入力するのは（当時の名残で）面倒です。

ところが近年、エディタ (VScode 等) やフォントが進化してプログラミングで使う特殊な記号列を**合字(ligature)**として処理できるようになりました。初心者が使うと逆に混乱すると思うので今の段階では強くおすすめはしませんが、興味のある人は新し目のプログラミング向けフォントをインストールして使ってみると良いでしょう。

以下は完全に同一の Python コードをフォントだけ切り替えて表示した例です。右がリガチャ対応フォント。(3 つの合字が使われているのがわかります？)

```
if a != 0 and a <= b:
    print(c := b / a)
```

```
if a ≠ 0 and a ≦ b:
    print(c := b / a)
```

練習問題

問:

- 文字列を入力し、
- それが**OITであればGood**、
- それが**OITでなければBad**、
- さらに最後に (OIT であろうがなかろうが)**end**と出力する

プログラム **isoit.py** を作成せよ。

(注) 本来は**else**を使うのが（この場合は）あるべき姿です。**else**は次回やります。

練習問題解答例

練習問題解答例

```
s = input()
if s == "OIT":
    print("Good")
if s != "OIT":
    print("Bad")
print("end")
```

Python のインデントは地雷原につき注意！

インデントは……

- うっかり欠落しやすい。
- うっかり挿入しやすい。
- 狂ってもエラーになりにくい。

インデント恐怖体験が次ページに……。

インデント地雷踏みました。

朝寝坊したらどうなるでしょう？

1. 目覚める。
2. 時刻を確認する。
3. **もし**時刻が 8 時より早ければ、
4. 　　顔を洗う。
5. 　　朝ご飯を食べる。
6. 　　歯磨きをする。
7. 　　着替える。
8. 大学に行く。

```
# 本当に動く Python プログラム
import time
print('目覚めました')
t = time.localtime()
if t.tm_hour < 8:
    print('顔を洗いました')
    print('朝ご飯を食べました')
    print('歯磨きしました')
    print('着替えました')
print('出発しました')
```

ちなみにこれはふつうにエラー。

見た目はなかなかオシャレですけどね。

1. 目覚める。
2. 時刻を確認する。
3. もし時刻が 8 時より早ければ、
4. 顔を洗う。
5. 朝ご飯を食べる。
6. 歯磨きをする。
7. 着替える。
8. 大学に行く。

```
# 本当に動かない Python プログラム
import time
print('目覚めました')
t = time.localtime()
if t.tm_hour < 8:
  print('顔を洗いました')
    print('朝ご飯を食べました')
      print('歯磨きしました')
print('着替えました')
print('出発しました')
```

1 つのカタマリ（suite）のインデントは一定にすべし。

コラム: `if`のうしろに続く式

授業が簡単すぎて退屈な人向けの暇つぶし読み物 (本ページは授業中に解説しません。)

- この授業では『**`if` 条件の式 `:`**』としました。**条件の式** とは『1 つ以上の**比較** (comparison) を 0 個以上の (今回のもう少しあとでやる) **`and`**, **`or`**でつなげたもの』です。で、比較というのがいわゆる関係演算子たち (**`<`**, **`>`**, **`<=`**, **`>=`**, **`==`**, **`!=`**) および **`not`** からなる式です。条件の式の評価結果は**`True`**もしくは**`False`**になります。ここまではまあ常識の範囲であまり難しいこと考えずに使えます。この授業の **`if`** も基本的にほぼここまでです。
- ただ本当は**`if`**のうしろは**必ずしも条件の式である必要はありません**。他の多くのプログラミング言語同様に Python でも**`if`**のうしろは一般の式 (expression) が書けます。式の評価結果 (計算した結果の値) は、数値だったり文字列だったり様々です。
- さて**`True`**, **`False`**以外のときの **`if`** の振る舞いはどうなるでしょう? 実はこれが**言語によってけっこうバラバラ**です。例えば**`0`**は Python だと**`False`**扱いですが、Ruby だと**`true`**扱いです。**`True`**, **`False`** 以外はエラーという言語もあります。新しい言語を学ぶときはその辺も要チェック。

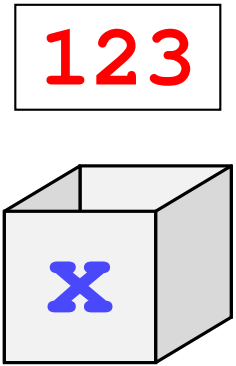
- ここから数ページは「難しくはないけれど、何が言いたいのかわからない」と感じると思います。
- 実際この授業の範囲ではさほど役に立たない話です。
- ですが、高いプログラミング技術を習得したいと思う人はしっかりと意識しておくといい、とてもためになるお話です。

変数に抱いているイメージは？

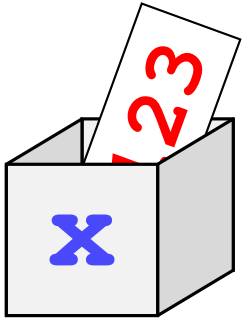
```
x = 123
```

この**代入のイメージ**を頭の中に描いてください。

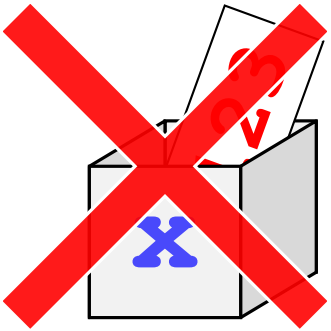
ひょっとして、こんなの?



ひょっとして、こんなの?



ひょっとして、こんなの?

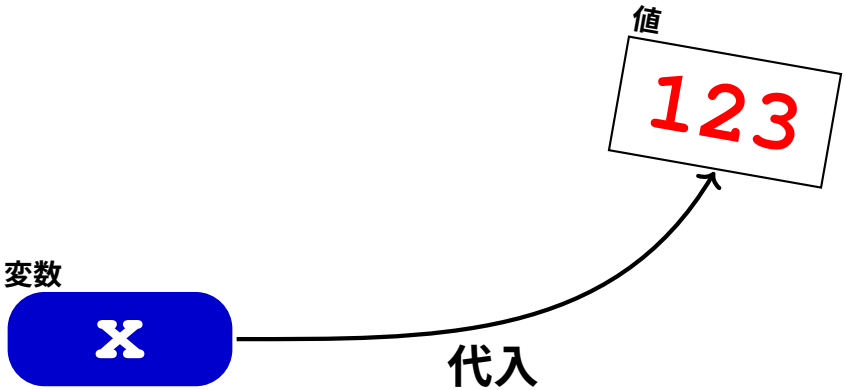


{初心者のうち, 言語によって} はこれでも良いが、Python でいつまでもこのイメージを引きずるとやがてハマる。

できればさっさとこっちのイメージに切り替えよう！

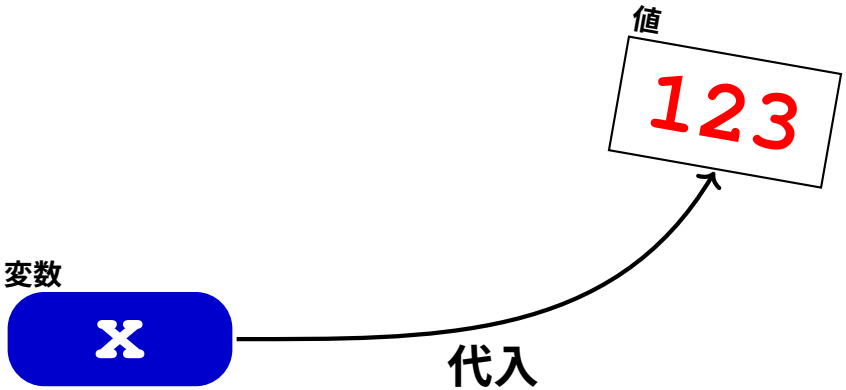
変数とは　　なり。

できればさっさとこっちのイメージに切り替えよう！



変数とは　　なり。

できればさっさとこっちのイメージに切り替えよう！



変数とは 値に対する**ラベル**あるいは**名前** なり。

練習問題と課題

練習問題 (自己チェッカーの「ヨ」でチェックできます。)

問: 『**幅 30 文字、高さ 7 文字の「ヨ」の字**』を**A**で書くプログラムを作成せよ。ただしコード中の '**A**' の文字は 10 個以下とすること。また全角文字は使わないこと。

実行例:

```
hkoba@host:~/prog1$ python3 yo.py
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                              A
                              A
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                              A
                              A
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

練習問題 (自己チェッカーの「ヨ」でチェックできます。)

問: 『**幅 30 文字、高さ 7 文字の「ヨ」の字**』を**A**で書くプログラムを作成せよ。ただしコード中の '**A**' の文字は 10 個以下とすること。また全角文字は使わないこと。

実行例:

```
hkoba@host:~/prog1$ python3 yo.py
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                              A
                              A
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                              A
                              A
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

解答例:

```
print("A" * 30)
print(" " * 29 + "A")
print(" " * 29 + "A")
print("A" * 30)
print(" " * 29 + "A")
print(" " * 29 + "A")
print("A" * 30)
```

練習問題

問: 西暦を入力すると令和に変換して出力するプログラム**ad2wa.py**を作れ。**2024**と入力したら単に**6**と表示すればいい。

実行例:

```
hkoba@host:~/prog1$ python3 ad2wa.py
2024
6
```

練習問題

問: 西暦を入力すると令和に変換して出力するプログラム**ad2wa.py**を作れ。**2024**と入力したら単に**6**と表示すればいい。

実行例:

```
hkoba@host:~/prog1$ python3 ad2wa.py
2024
6
```

解答例:

```
ad = int(input())
wa = ad - 2018
print(wa)
```

練習問題〜ad2wa ver 2.0 〜

問: 明治から令和に対応した西暦 → 和暦変換プログラムを作れ。なお令和 1 年 = 2019 年、平成 1 年 = 1989 年、昭和 1 年 = 1926 年、大正 1 年 = 1912 年、明治 1 年 = 1868 年である。

実行例:

```
hkoba@host:~/prog1$ python3 ad2wa.py
1999    # <-入力
heisei  # -> 出力 1 行目
11      # -> 出力 2 行目
```

練習問題〜ad2wa ver 2.0 〜

問: 明治から令和に対応した西暦 → 和暦変換プログラムを作れ。なお令和 1 年 = 2019 年、平成 1 年 = 1989 年、昭和 1 年 = 1926 年、大正 1 年 = 1912 年、明治 1 年 = 1868 年である。

実行例:

```
hkoba@host:~/prog1$ python3 ad2wa.py
1999    # <-入力
heisei  # -> 出力 1 行目
11      # -> 出力 2 行目
```

解答例:

```
ad = int(input())
wa = ad - 2018
era = "reiwa"
if ad < 2019:
    wa = ad - 1988
    era = "heisei"
if ad < 1989:
    wa = ad - 1925
    era = "showa"
if ad < 1926:
    wa = ad - 1911
    era = "taisho"
if ad < 1912:
    wa = ad - 1867
    era = "meiji"
print(era)
print(wa)
```

練習問題 (自己チェッカーの「OIT」でチェックできます。)

問: 標準入力から文字列を一つ受け取り、それが**OIT**であれば**OIT_OIT_OIT_OIT_OIT**と出力し、それ以外であれば**bye**と出力するプログラム**isoit.py**を作成せよ。ただし、コード中に**OIT**という文字列は 1 度しか使ってはならない。

実行例:

```
hkoba@host:~/prog1$ python3 isoit.py # ←実行
OIT                                  # <-入力
OIT_OIT_OIT_OIT_OIT                  # <-出力
hkoba@host:~/prog1$ python3 isoit.py # ←再実行
IoT                                  # <-入力
bye                                  # <-出力
```

paiza ラーニング

`if`の初級中の初級問題はこちら ↓。

- 【paiza ラーニング】→ 【問題集】→【条件分岐メニュー】→【0 が含まれていないか判定】→STEP 1, 2, 4
- 【paiza ラーニング】→ 【問題集】→【D ランクレベルアップメニュー】→【占い】→STEP 1, 2, FINAL

課題: 以下の仕様のプログラムを作成せよ。

実行例:

```
user@host:~/prog1$ python3 rbox.py
16      <-- 数値を入力
B       <-- 1 文字の文字列を入力
 BBBBBBBBBBBBBB
B              B
B              B
B              B
 BBBBBBBBBBBBBB
user@host:~/prog1$
```

- 最初に**整数値**(例: 16) を入力し、
- 次に**1 文字の文字列**(例: B) を入力すると、
- 幅がその整数値で高さが 5 文字の**角の取れた四角形**を出力するプログラムを作成せよ。ただし、**整数値が 10 未満の場合は幅を 10、31 以上の場合は幅を 30 にする**こと。
- 実行例は下記。数値と文字によって結果がきちんと変わるように。
- 部分的にできていれば部分点がつくので無理せず少しずつ実装しよう。

第 4 回 (4/30) 課題 (提出期間: 4/30〜5/6)

- 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/Rgo7xAzsRnUcaGqVA`