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。

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

8 (前半) of 14

a LaTeX + Beamer slideshow

復習の練習: ぎゃくくく (4 分)

問. 整数値を**一つ**入力すると、その『段』の九九を**逆に** 出力するプログラム**rev99.py**を作成せよ。

実行例

```
7   ← 7の段(入力)
===
7
9
63
===
7
8
56
```

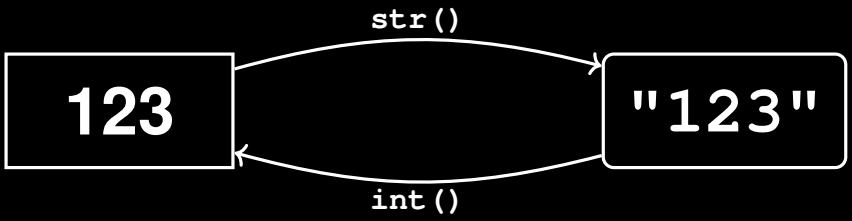
```
===
7
7
49
===
7
6
42
===
```

```
7
5
35
===
(中略)
===
7
1
7
```

本日のお題

- 前ページの出力をもう少し見栄え良くしたい。
- ようは数値と文字列を並べて出力したい。(『**7*6=42**』みたいに。)
- さてどうするか？ (方法はいろいろあるけど**応用範囲の広いやつ**希望)

型変換



(参考) print の便利機能

入門書の類の多くのものに割と使われているので知っている人も多いかも。

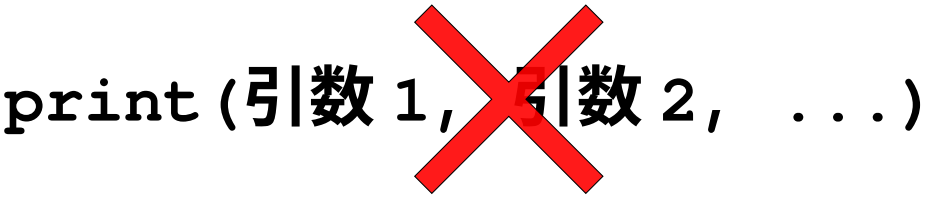
print(引数 1, 引数 2, ...)

print に複数の引数を与えると、それらを
いい感じに横に並べて出力してくれる。

```
w = int(input())
y = w + 2018
print("A.D. ", y)
```

`print`の便利機能を封印してみた

これは“応用範囲の広いやつ”ではまったくない＆一生知らなくても大して困らない。



Python の print 関数の可変長引数を diss る

この“便利機能”は（複雑な文法の上に成り立っているために）初心者の**文法理解を妨害**し、工夫や努力させず、**堕落させてしまう**……。ついでにマトモな**代替手段はいくらでもある**。

復習のクイズ

```
name = "Kobayashi"
print("NAME=")
print(name)
```

一行で出力するには？

文字列と文字列の足し算

```
name = "Kobayashi"
print("NAME=" + name)
```

文字列と文字列は+でつなげられる。

文字列と数値をつなげたい。

```
sc = 80
print("SCORE=")
print(sc)
```

一行で出力するには？

これは無理。

```
sc = 80
print("SCORE=" + sc) # エラー
```

文字列と数値は+演算ができない。

敗因と対策

敗因

```
print("SCORE=" + sc)
```

文字列と数値は足し算できない。

対策

```
print("SCORE=" + 《文字列変換した sc》)
```

数値を文字列に変換すればいける…か!?

str 関数

文字列に変換する

結論

`str(数値)` とすると**数値を表す文字列**が得られる。

さあ↓これはどうすればいい？

```
print("SCORE=" + sc) # エラー
```

やっと、問題解決

```
print( "SCORE=" + sc )
```

文字列　　数値

```
print( "SCORE=" + str(sc) )
```

文字列　　文字列

練習: ぎゃくくく ver. 2 (4 分)

問. `rev99.py`を改変し、整数値を**一つ**入力すると、その『段』の九九を**逆に** 以下の実行例のように出力するプログラムを作成せよ。(難易度: ★★☆☆☆)

実行例

```
7   ← 7の段(入力)
7*9=63
7*8=56
7*7=49
7*6=42
(続く)
```

```
(続き)
7*5=35
7*4=28
7*3=21
7*2=14
7*1=7
```

練習: 九九表 (5 分)

問. 整数値を 2 つ (a, b とする。) 入力すると、a の段から b の段までの九九表を出力するプログラム**table99.py**を**二重ループで**作成せよ。

実行例

```
3    ← 3の段から(入力)
6    ← 6の段まで(入力)
 3   6   9 12 15 18 21 24 27
 4   8 12 16 20 24 28 32 36
 5 10 15 20 25 30 35 40 45
 6 12 18 24 30 36 42 48 54
```

- **レベル 1** (難易度: ★★★★☆)
  とりあえず縦横に並んだ九九が表示される。
- **レベル 2** (難易度: ★★★★★)
  左の実行例のように縦の並びが崩れずに (値が 1 桁と 2 桁の場合とで間隔が調整されるように) 表示される。
- ヒント: 《段》をいきなり print しようとせず、まず文字列として作る。

`int` 関数

文字列を整数値にするint関数

結論:
int関数は**str**関数とだいたい逆の働きをする。つまり、文字列を (整) 数値と解釈した値が得られる。

クイズ: 何が出力されるかな?

```
s = str(123)
x = int("456")
print(s * 3)
print(x * 3)
```

これもそういうことだった！

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

補足

- **str**はだいたい期待どおりやってくれる。ただし状況によっては末尾に『**.0**』がつく。
- **int**は**変換できない文字列はエラー**となる。
- **int**は**整数専門**。小数点以下があるとエラー。

```
print(str(6 * 2))
print(str(6 / 2))
print(int("123 USD"))
print(int("123"))
print(int("123.45"))
```

練習 (4 分)

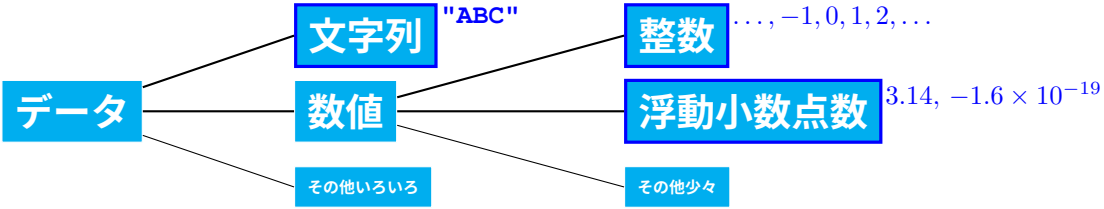
問. 2 桁の整数 x を入力すると、『x **を 3 度繰り返した数を 481 で割って、2 倍して、最初の数で割った数**』を出力するプログラム`stepcalcs.py`を作成せよ。(難易度: ★★★★☆)

例:

1. `45`を入力すると、
2. 3 度繰り返した数は **454545** で、
3. 481 で割ると（この場合ちょうど割り切れて）**945**。
4. それを 2 倍すると **1890**。
5. それを最初の 45 で割ると (またまたちょうど割り切れて)、**42** なので、
6. `42`と出力する。(注: この場合ふつうに作ると `42.0` と出力されるがそこは気にしなくていい。)

コラム: Python のデータの型

Python にはいくつかのデータの種類 (型) があり、型によって演算のルールが違います。…とここまでは既に何度もした話。型はざっくり **数値**と**文字列** に分けられます。本当はもっといっぱいあるのですが初級レベルで意識しなければならないものはとりあえずこの２つ。そして、数値はさらに**整数**と**浮動小数点数** に分けられます。(これも本当はもう少しあるのですが、気にしなくていい。) 今日学んだ**`int`**は文字列から整数を作る関数です。



数値と文字列は全然別モノですから使い方に迷うことはないと思います。さて問題です。整数って何か意味あるんでしょうか? ふつうに考えればよりさまざまな値を表せる浮動小数点数の方が高性能で、これだけあれば十分な気もしますよね? ところがさにあらず。**どちらでも良いときは整数**を使うのが鉄則です。理由は…紙面が尽きたので省略。

float 関数

重要な認識

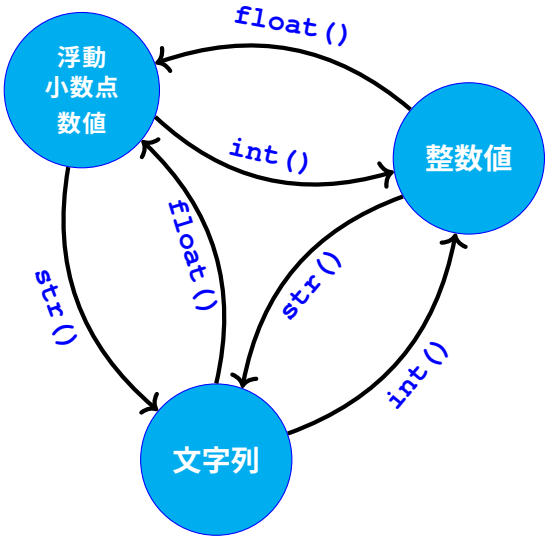
”123” と 123 は違うが、
123 と 123.0 も違う。

型の違いでこういうことも起こり得る、の例

```
# intfloat.py
# 末尾の ".0" をつけたり消したりしてみよう。
a = 10000000000000000.0
if a - 1 != a:
    print("Okay.")
else:
    print("What!?")
```

この結果そのものは Python の実装上の「たまたま」なのであまり重要ではないが、**整数**と**浮動小数点数**という二つの**型**があり、処理の結果が変わることがあるということを認識しておこう。

int, float, str 関数




```
# int() 関数の例
i1 = int("42")       # ok (42)
i2 = int(42.195)     # ok (42)
i3 = int("42.195") # error
```

```
# float() 関数の例
f1 = float("42")       # ok (42.0)
f2 = float(42)         # ok (42.0)
f3 = float("42.195") # ok (42.195)
```

```
# str() 関数の例
s1 = str(42)       # ok ("42")
s2 = str(42.0)     # ok ("42.0")
s3 = str(42.195) # ok ("42.195")
```

float関数の使いどころ

- 標準入力から浮動小数点数を入力したいときなど、**文字列を変換するとき**によく使う。
- それ以外では、（特に ver 3 以降の Python では）正直なところ**あまりない**。

```
# 円の半径を入力すると面積を出力するプログラム
print("radius=?")
r = float(input()) # <- int(input()) より適切でしょ？
print("Its area: " + str(r * r * 3.14))
```

補足: Python の型と四則演算

整数 (int)と**浮動小数点数 (float)**と**文字列 (str)**が四則演算によってそれぞれどのような値を生じるのかをまとめました。実は python の数値にはもう少し他のものもあるのですが、とりあえずざっくりこの 2 種類があるんだと思っていてください。

+	整	浮	字
整	整	浮	err
浮	浮	浮	err
字	err	err	字

-	整	浮	字
整	整	浮	err
浮	浮	浮	err
字	err	err	err

*	整	浮	字
整	整	浮	字
浮	浮	浮	err
字	字	err	err

/	整	浮	字
整	浮	浮	err
浮	浮	浮	err
字	err	err	err

bool 型

0 とか 1 とか意味わからん

簡単な自動運転車両の制御プログラムを考えよう。

- **稼働中** (**in_service**) に、
- **人が乗って** (**aboard**) いて、
- **障害物** (**obstacle**) がなければ

前進 (**forward()**) する、というプログラムを考える。

```
if in_service == 1 and aboard == 1 and obstacle == 0:
    forward()
```

これら 3 つの変数は**フラグ**のつもり。……でもこれでいいのか？

- **in_service** は本当に “1” が稼働中？
- **aboard** ってもしかして乗車している人数かな？
- **obstacle** は障害物までの距離を表しているのでは？

などなど、コードを**読む人が不安になる**。

【bool型】降臨

- **整数型** (`int`) は正負に無限の値を取れる。
- **文字列型** (`str`) は無限のパタンの文字列を表せる。
- **浮動小数点数型** (`float`) は正負のいろいろな値を取れる。

`bool`型は`True`と`False`の 2 値のみ！

意味が明確になるので yes/no 的な**フラグには bool 型**を使うと良い。

でもちょっと具体的にイメージわかない、と思うので次ページで例示。

bool 型で『フラグ感』を明確にした素数判定

```
x = int(input())
is_prime = True  # 素数かどうか？を表すフラグ。(もちろん True なら素数。)
if x < 2:
    is_prime = False
else:
    i = 2
    while i < x:
        if x % i == 0:
            is_prime = False
            break
        i = i + 1
if is_prime == True:    # この行については次ページで補足
    print("a prime number!")
else:
    print("not a prime number...")
```

実は……。(その 1)

ちょっと難しいのでわからなかったらスルーしてください。

関係演算子を使った演算の結果は、実は `bool` 型だった!

```
# python の REPL
>>> 1 + 2 == 3
True
>>> "OIT" == "IoT"
False
>>> 4 > 5
False
>>> is_prime = True != False
>>> is_prime == True
True
```

実は……。(その 2)

ちょっと難しいのでわからなかったらスルーしてください。

`if` や `while` は、続く値を `bool` 型であるとみなして処理を切り替えていた!

そんなわけで、bool 値のフラグのチェックは、こんなふうに書ける。

```
# 素数判定プログラムから抜粋
if is_prime:      # "== True"は不要
    print("a prime number")
```

```
# 自動運転車両プログラムはフラグが bool 型ならこう書ける
if in_service and aboard and not obstacle:
    forward()
```

実は……。(その 3)

細かいことなのですが、理解できる人は理解してきましょう。

- 数値の場合、**0 は False 扱い**、**0 以外は True 扱い** される！
- 文字列の場合、**空文字列 False 扱い**、**それ以外は True 扱い** される！

ただ、これ (数値や文字列を条件式の値として使うこと) は C 言語時代からの名残りのようなもので、使っても良いことはほとんどないから、できるだけ使わないようにすることを（小林は）推奨します。bool にすべきところは、きちっと True か False の値になる式を書く。

```
a = int(input())
if a: # こういうこともできることはできる（非推奨。a != 0 と書きましょう)。
  print("It's a non-zero value, right?")
```

課題: 以下の仕様のプログラム (九九練習帖) を作成せよ。

- 最初に**問題数**を入力する。
- 1〜9 のランダムな 2 つの数からなるかけ算の練習問題を入力した問題数個 (1 行に 1 問) 出力する。
- 表示は "**6*7=**" のようにすること。(数値、*、= の間にスペースを入れない)
- 答えは出力しなくていい。
- 以上で 2 点。さらに解が 9 未満の問題が出ないようになっていれば 3 点。

実行例 (毎回変わること)

```
5          # ←入力: 問題数
3*5=       # →出力: 1問目
9*1=       # →出力: 2問目
6*7=       # →出力: 3問目
2*5=       # →出力: 4問目
8*9=       # →出力: 5問目
```

難易度: ★★★★☆

第 8 回 (6/11) 課題 (提出期間: 6/11〜6/18)

- **絶対に、独力でやること。**逆に誰かに聞かれても教えないこと。
- 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`

(またやります)【ウデマエ自慢】短いコードが書けた人は`prog1@oitech.ddns.net`まで！優秀なコードは表彰します。(名前を出して欲しくない人はひと言その旨書いておいてください。)