Python 講座

【第 1 回 #1】2015.04.28

この講座は……

- 対象：プログラミング初心者（～中級者）
- やること
  - 授業の復習＋概念的な話＋掘り下げた話
  - 授業の予習
    - 初回なので、今回は復習で終わるかも……

授業はどこらへん？

1	導入、入出力処理
2	値と変数
3	条件分岐
4	繰り返し (今週)
5	第1回から4回の総括
6	関数
7	ファイル操作
8	対話ループ
9	複合データ
10	第6回から9回の総括

11	応用(1) タートルグラフィックスで幾何学模様
12	応用(2) Tkを利用したレンダリング
13	総括(1) 内容の復習と実際的なプログラムの作成
14	総括(2) 実際的なプログラムの作成
15	まとめ

- 最後にやったのは if とか elif とか else
  - シラバス通りならそのはず……

きょうの話

1. 変数と入出力
2. 数値の演算と文字列の操作
3. リストと辞書

▼ 時間があれば話す ▼

4. 真偽値とその演算、条件分岐

変数と入出力

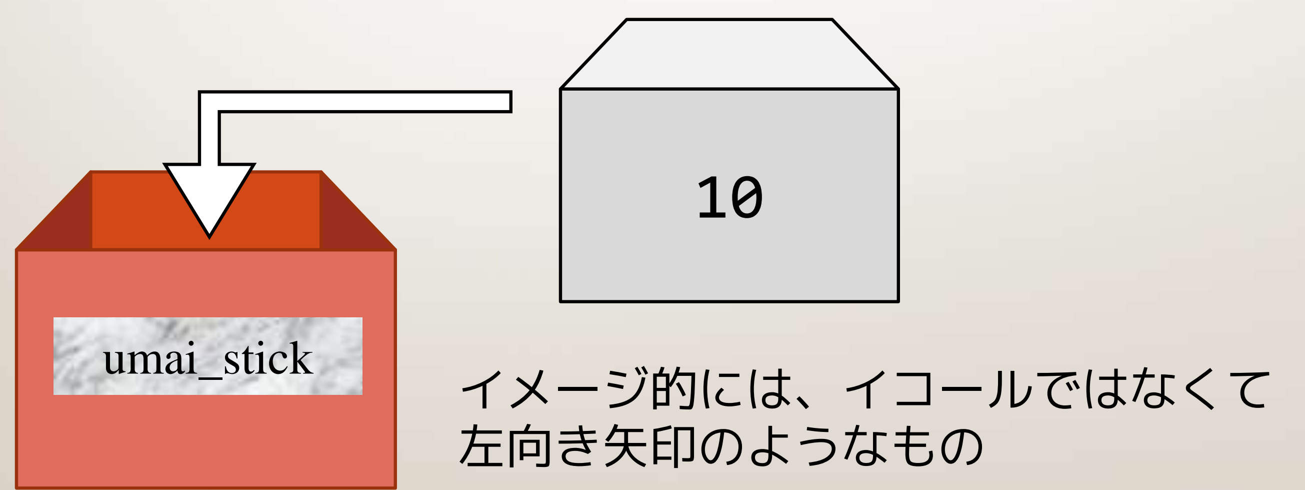
print(‘hoge’) とか、fuga = input()

変数？

- いろんな値を保持してくれる入れ物のこと
- 値を入れる（代入する）には？
  - *<変数名>*に続けて、＝*<値>*

```
umai_stick = 10⏎
```

変数 umai_stick に 10 を「代入」



イメージ的には、イコールではなくて
左向き矢印のようなもの

リテラルいろいろ

```
hoge = 10⏎
```

整数として解釈されるコードの一部を**「整数リテラル」**という。

```
hoge = 10.0⏎
```

実数として解釈されるコードの一部を**「実数リテラル」**という。

小数点の有無で決まる

```
hoge = '10'⏎
```

文字列として(ry
「文字列リテラル」という。

クォーテーションで囲むと文字列

文字列リテラルいろいろ

円記号￥がバックスラッシュ＼で
表示されるが、気にしない。

```
'I¥'m lovin¥' it'⏎
```

文字列中でクォーテーションを
使うときは、頭に￥を置く。

```
'暇を¥n持て余した¥n神々の遊び'⏎
```

改行を入れたいときは、¥n を使う。もしくは、

```
'''暇を⏎
持て余した⏎
神々の遊び'''⏎
```

クォーテーション 3 個で囲んで、
直接 [Enter] してもいい。

※改行は後述する print を使うと反映されます

Advanced

変数の名前

余裕があれば
読むといいかも。

- アルファベット、数字、アンダーバーで名づける
  - １文字目に数字は使えません！
- 変数名にできない単語もある
  - Python の言語内で特別な機能を持つ単語は使えない
  - class 、continue 、for 、try などなど……
- 自分なりの法則で名前を付けると吉
  - 変数名は全て小文字＆単語間にアンダーバーとすることが多い。例: student_id

画面に表示する

- print(★)
  - ★ …… 表示（出力）したい内容。省略可。
  - print 関数。指定した内容 + 改行を表示してくれる。

```
print(123)⏎
```
「123」と表示する。

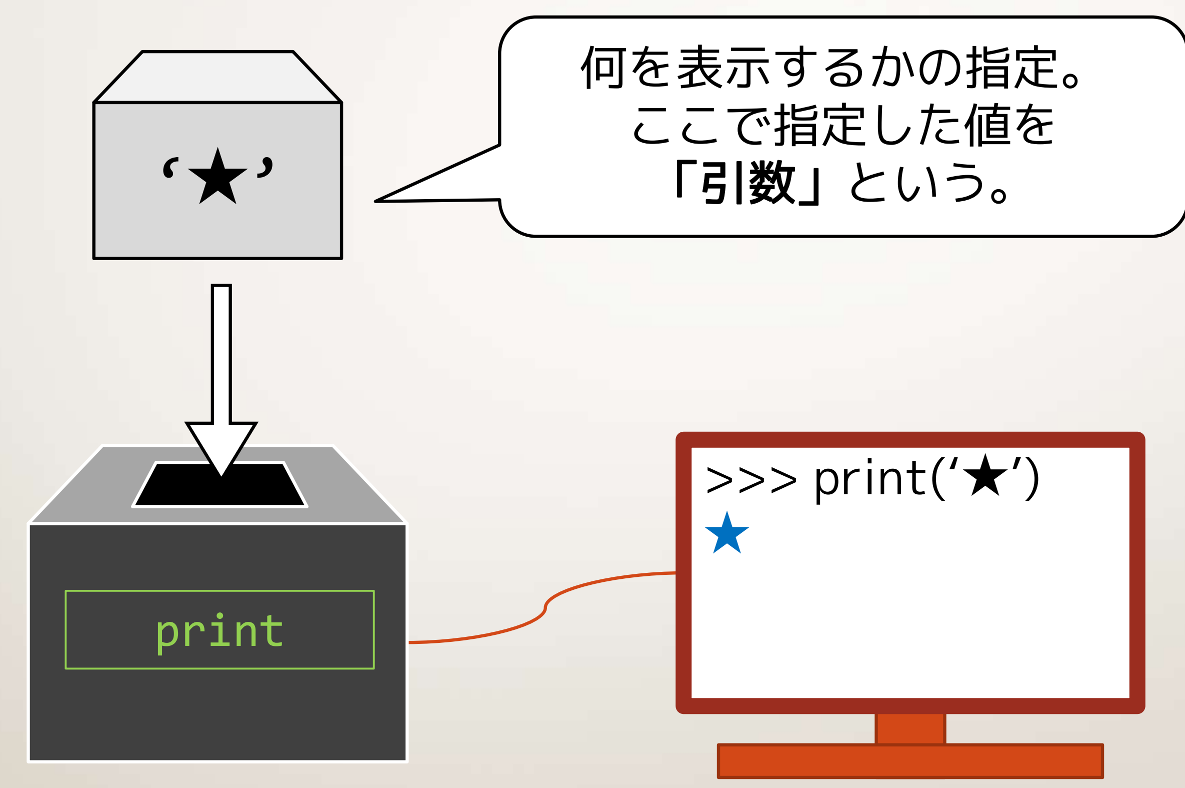
```
print(123.456)⏎
```
「123.456」と表示する。

```
print(umai_stick)⏎
```
変数 umai_stick の中身を表示する。

```
print('umai_stick')⏎
```
「umai_stick」と表示する。

やっぱり文字列はクォーテーションで囲む

print()

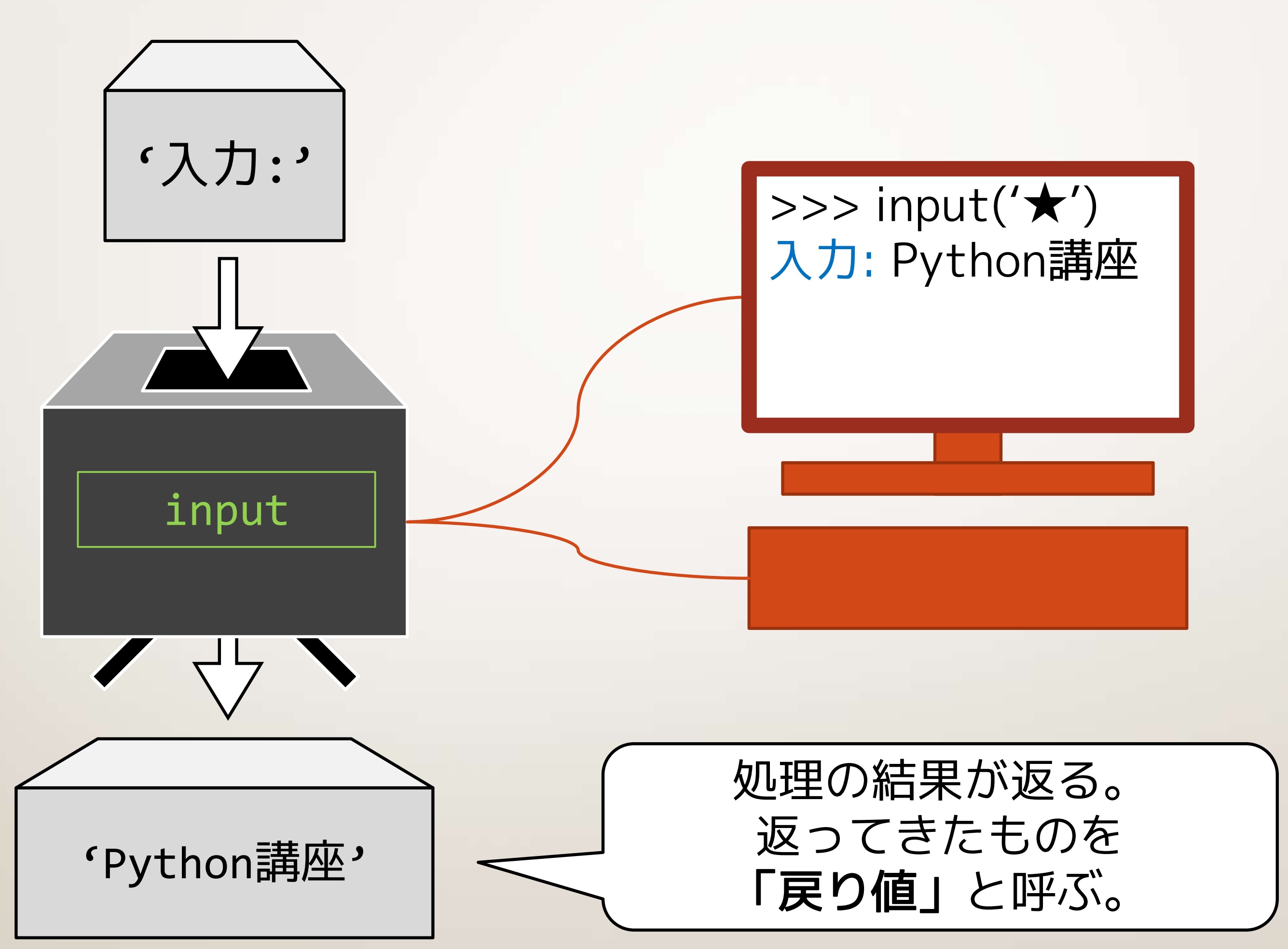


入力を受け取る

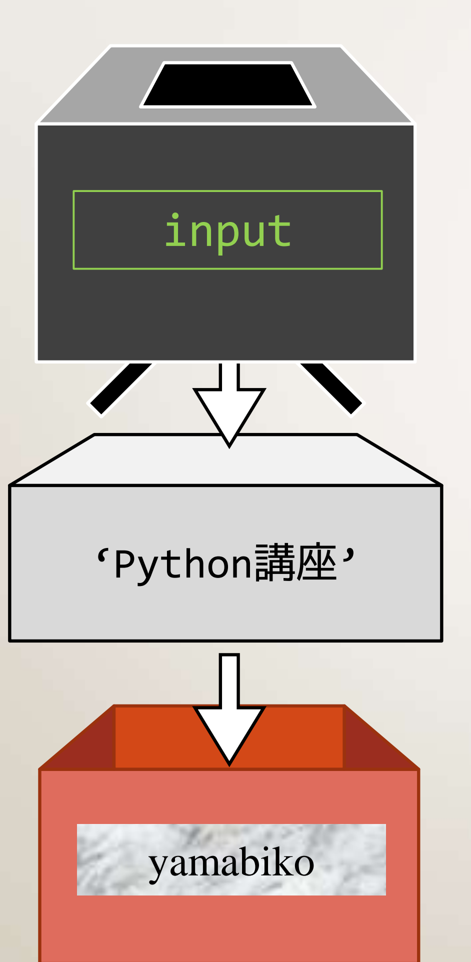
- input(★)
  - ★ …… 画面に表示する内容。省略可。
  - input 関数。ユーザーからの入力を文字列で返す。

```
yamabiko = input('入力してください: ')⏎
print(yamabiko)⏎
```

input()



入力を受け付けて保持

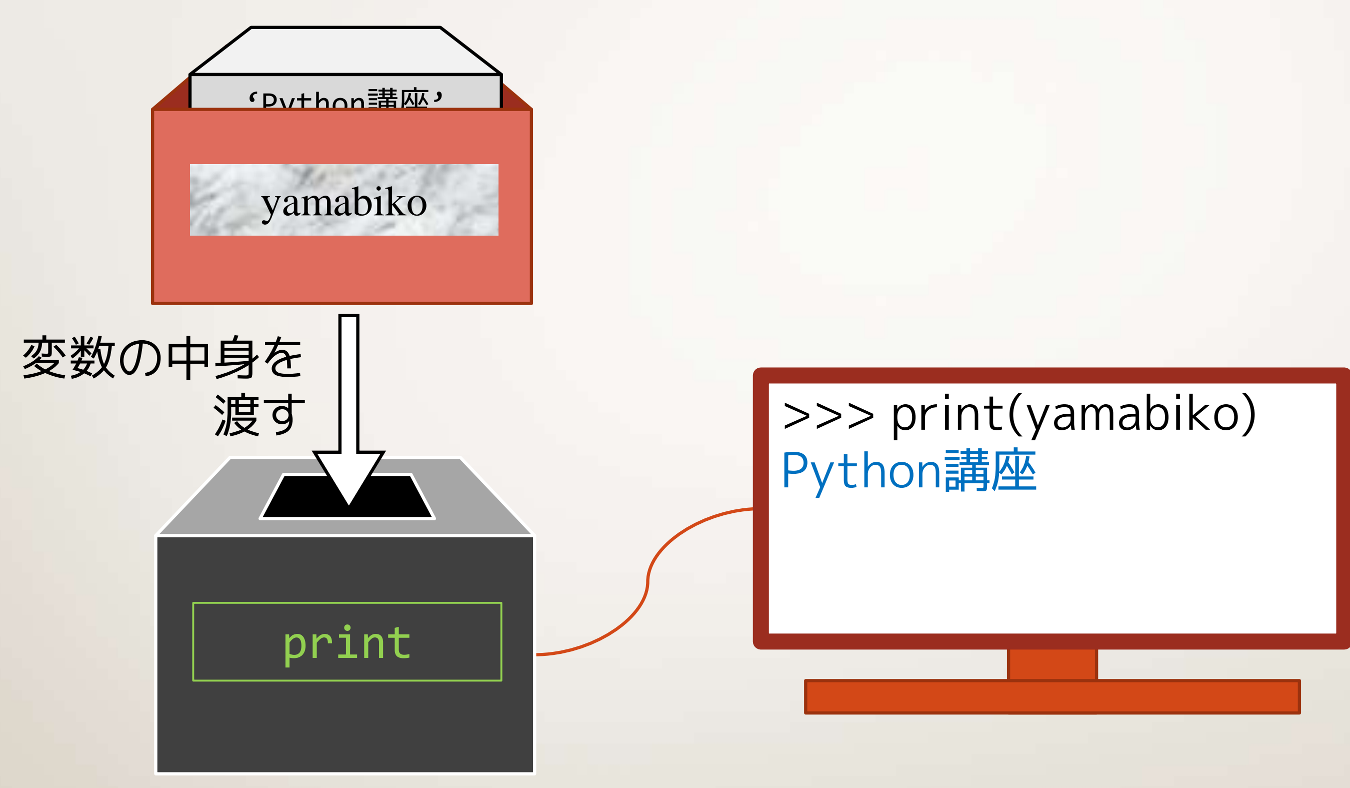



```
yamabiko = input(‘入力求む:’)⏎
```

input(★) の部分が具体的な
文字列に置き換わるようなもの

```
yamabiko = ‘Python講座’⏎
```

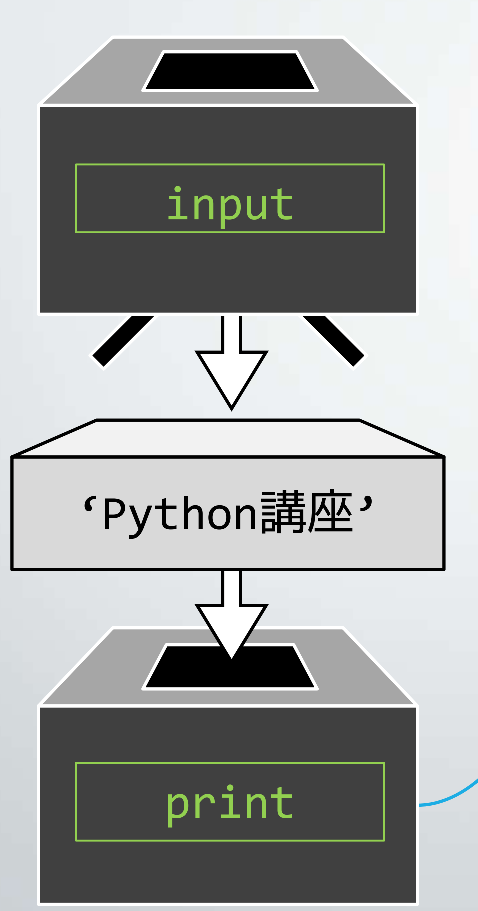
保持した値の表示




```
yamabiko = input('入力求む')⏎
print(yamabiko)⏎
```

Advanced

変数を使わないVer.



print で表示する内容を、
input の結果として指定

```
>>> print(input('入力: '))
入力: Python講座
Python講座
```

ここまでOK？

- 変数の作り方
  - 作り方だけ。中身のいじり方は次でお話します
- 入力を受け取るには input 関数
- 画面に出力するには print 関数

数値の演算と文字列の操作

+ - * / % と、format の話

数値と演算子 (1/2)

- 演算子とは、+ やら * やら // のこと。

```
13 + 5⏎
18
```

2 値を足した値。

```
13 - 5⏎
8
```

左から右を引いた値。

```
13 * 5⏎
65
```

2 値を掛けた値。

```
13 / 5⏎
2.6
```

左を右で割った値。

```
13 % 5⏎
3
```

左を右で割った余り。

数値と演算子 (2/2)

- 計算順序は数学と同じ
  - 左から右に進む
  - 掛け算と割り算は、足し算と引き算よりも先
    - カッコ () で囲んで演算順をコントロール

```
1 * (2 + 3) - (4 + 5) / 6⏎
3.5
```

```
1 * ((2 + 3) - (4 + 5)) / 6⏎
-0.6666666666666666
```

Advanced

演算子 +alpha

- べき乗もできる。
  - 演算順序は掛け算・割り算よりも先

```
1 / 10 ** 2⏎
0.01
```

「10 の 2 乗」分の 1。

- 小数以下を切り捨てる割り算もできる。

```
10 // 4⏎
2
```

10 ÷ 4 = 2.5
小数以下を切り捨てて 2 。

```
10 // 4 / 3⏎
0.6666666666666666
```

10 // 4 = 2 なので、
2 ÷ 3 と等価。

変数と演算子

- 先に述べた演算は、変数に対しても行える。

```
umai_stick = 10⏎
umai_stick * 3⏎
30
```

変数 umai_stick の値に 3 を掛ける場合。

- 演算結果は「値」なので、いろんなことに使える。

```
price = umai_stick * 3
```

変数に代入する中身として

```
print(umai_stick * 3)
```

関数に指定する内容として

umai_stick * 3 を 30 に置き換えても結果は等価

Advanced

複合代入演算子

- 計算と代入を同時に行う演算子もある。
  - 下記のほか、-=　*=　/=　**= も使える

```
a += 3⏎
```

変数 a の中身に 3 を足す。
a = a + 3 と等価。

```
a //= 3⏎
```

変数 a の中身を 3 で割り、少数切り捨てする。
a = a // 3 と等価。

```
a %= 3⏎
```

変数 a の中身を 3 で割った余りを代入する。
a = a % 3 と等価。

※代入先の変数（この場合は a ）が存在して、なおかつ中に数値が入っていないとエラーになる

文字列と演算子

- 文字列に使える演算子は、+ と *
  - 引き算・割り算は存在しないので注意
  - 数値同様、掛け算が優先されます。

```
‘kitsune’ + ‘mimi’⏎
‘kitsunemimi’
```

2 つを結合した文字列。文字列同士なら OK。

```
‘con’ * 3⏎
conconcon
```

指定回数だけ繰り返した文字列。整数なら OK 。

変数と演算子

- やっぱり変数に対しても行える。

```
asterisk = ‘*’⏎
asterisk * 5⏎
*****
```

変数 asterisk の値を 5 回繰り返す場合。

- やっぱり結果はいろんなことに使える。

```
fluffy = ‘mofu’ + ‘fuwa’
```

変数に代入する中身として

```
print(‘mofu’ + ‘fuwa’)
```

関数に指定する内容として

‘mofu’ + ‘fuwa’ を ‘mofufuwa’ に置き換えても結果は等価

Advanced

複合代入演算子

- やっぱり計算と代入を同時に行う演算子もある。

```
a  +=  'con'⏎
```

変数 a の中身に 3 をつなぐ。
a = a + 'con' と等価。

```
a  *=  3⏎
```

変数 a の中身を 3 回繰り返す。
a = a * 3 と等価。

※代入先の変数（この場合は a ）が存在して、なおかつ中に文字列が入っていないとエラーになる

数値 in 文字列

- 文字列と数値は足し算できない……
  - 足せるのは文字列同士だけ

```
power_level = 530000 ⏎
‘戦闘力は’ + power_level + ‘です’⏎
Can’t convert ‘int’ object……
```

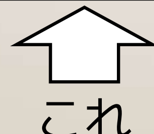
- どうする？
  - format を使って文字列の中に埋め込む

format？

- ★.format(☆, ☆, ...)
  - format メソッド。★ の中に ☆ で指定したものを埋め込んだ文字列を返す。
  - ★ …… ひな形となる文字列。
  - ☆ …… 埋め込む内容。カンマ区切りで複数指定可能。

- ひな形
  - 埋め込まれる場所を {} にしておく。複数設置可能。
  - {} のことをプレースホルダーと呼ぶ。

```
'戦闘力は{}です'⏎
```



format 使用例

```
print(‘戦闘力は{}です’.format(530000)) ⏎
戦闘力は530000です
```

```
hinagata = ‘戦闘力は{}です’ ⏎
power_level = 530000 ⏎
print(hinagata.format(power_level)) ⏎
戦闘力は530000です
```

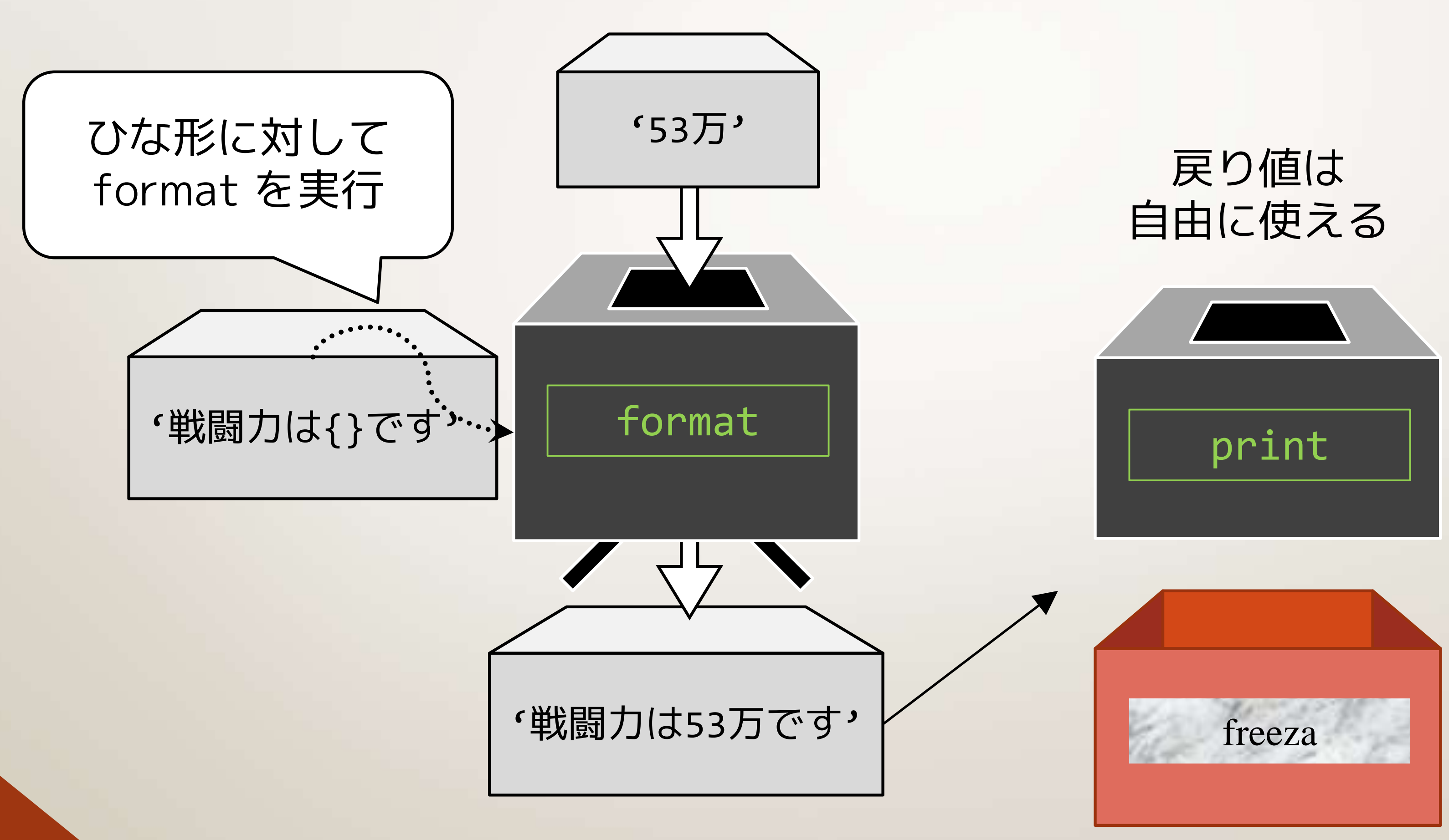
★ や ☆ は変数でも OK

数値以外も埋め込める

```
freeza = ‘戦闘力は{}です’.format(‘53万’) ⏎
print(freeza) ⏎
戦闘力は53万です
```

埋め込み済みの文字列は「返される」

format()



文字列と数値

- 見た目が同じでも、数値と文字列は区別される
  - 123 は数値
  - ‘123’ は文字列
    - 123 + 123 はできるけど、’123’ + 123 はできない。
- 例えば input() で「123」を入力しても、文字列として扱われる
  - 文字列を数値として扱うには「変換」が必要

文字列と数値の相互変換 (1/2)

- int(★)
  - ★ に指定された文字列や実数を「整数」にした値を返す。
    - 実数は、小数以下が切り捨てられる
  - 注: 「実数の文字列」はエラーになる。'123.45' など。

- float(★)
  - ★ に指定された文字列や整数を「実数」にした値を返す。

- str(★)
  - ★ に指定されたものを「文字列」にした値を返す。

文字列と数値の相互変換 (2/2)

```
'123' + 123 ⏎
Can't convert 'int' object...
```

```
int('123') + 123 ⏎
246
```

文字列の '123' は、
int() で変換すると
整数として扱える
ようになる。

```
int('123.45') ⏎
invalid literal for int()...
```

```
int(float('123.45')) ⏎
123
```

文字列を整数にする
時は、まず float() で
実数にした上で、
int() に処理させる。

ここまで OK？

- 数値の演算子　　…… + - * / %
- 文字列の演算子　…… + *
  - それぞれ数値同士、文字列同士なら OK

- 文字列に埋め込むには ひな形.format(値)
  - {} は複数OK、値はカンマ区切りで複数OK

- int(★)　　…… ★ を整数にして返す
- float(★)　…… ★ を実数にして返す
- str(★)　　…… ★ を文字列にして返す

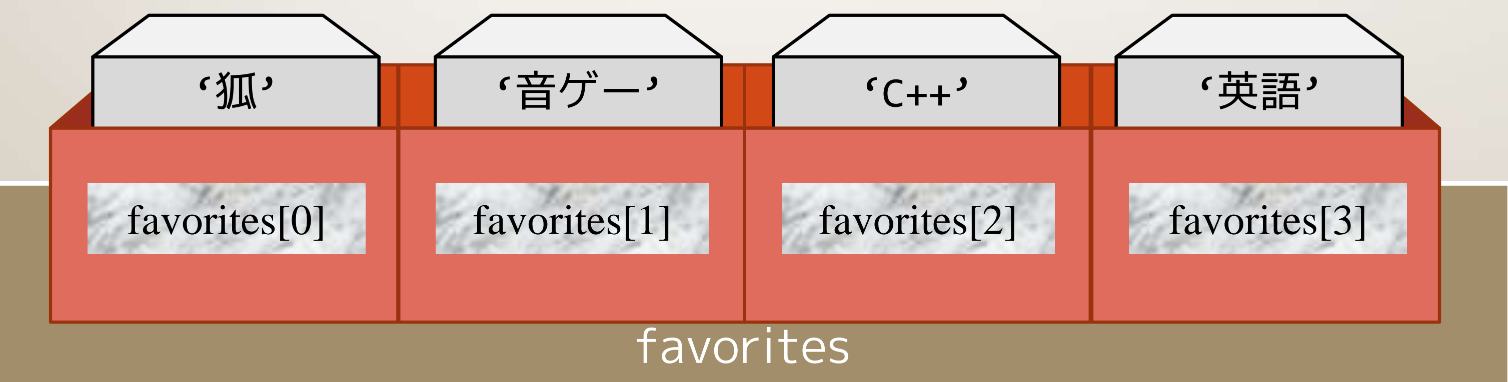
リストと辞書

[0, 1, 2] と {‘a’: ‘apple’, ‘b’: ‘banana’} の話

リスト？

- 値の「列」を表現するのに使う。
- 角カッコ [] で囲み、カンマ区切りで値を入れる。
  - 入る値は何でも OK、統一されていなくてもいい

```
favorites = [‘狐’, ‘音ゲー’, ‘C++’, ‘英語’]⏎
```

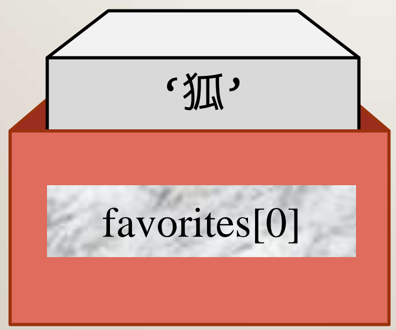


リストの使い方

- 要素（＝入っている値）を見に行くには……
  - *<リスト名>* [*<何番目>*]
  - 番号は 0 番からスタート; 末尾は (全要素数-1) 番目
  - 範囲外の番号はエラーになります

```
print(favorites[0])⏎
狐
```

```
print(favorites[4]) ⏎
IndexError...
```

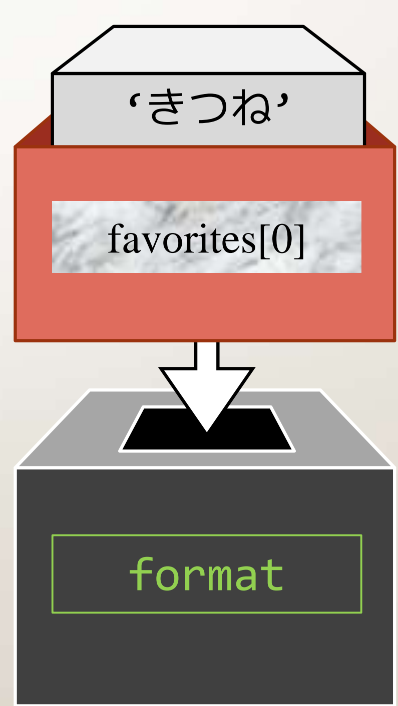


要素の使い方

- 変数と全く同じ！
  - *<リスト名>* [*<何番目>*] をひとかたまりとして扱おう

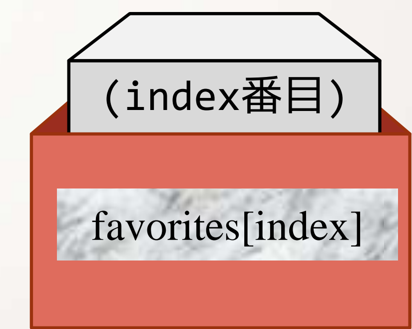
```
favorites[0] = ‘きつね’⏎

‘{}かわいい’.format(favorites[0])
きつねかわいい
```



リストの使い道 (1/2)

- リテラルあるところ、変数あり
  - *<何番目>* の部分は数値変数にもできる
  - 例えば、入力してもらった番号の値を画面に表示する、など




```
index = int(input('何番目? (0-3): ')) ⏎
print(favorites[index])⏎
```

```
何番目? (0-3): 1 ⏎
音ゲー
```

リストの使い道 (2/2)

- 数値がつまったリスト御用達
  - sum(★) ……要素の総和を返す
  - min(★) ……リスト中の最小値を返す
  - max(★) ……リスト中の最大値を返す
    - ★ …… リスト名

```
values = [1, 2, 4, 8, 16, 32, 64, 128]⏎
print(sum(values)) ⏎
255
print(min(values)) ⏎
1
print(max(values)) ⏎
128
```

リストの切り出し

- *<リスト名>[<ここから>:<ここまで+1>]*
  - リストの一部分だけ新たなリストをつくる。
  - 範囲に注意！終端+1 を指定します

```
sentence = [‘The’, ‘quick’, ‘brown’, ‘fox’] ⏎

print(sentence[2:4]) ⏎
[‘brown’, ‘fox’]

adjectives = sentence[1:3] ⏎
print(adjectives[0]) ⏎
quick
```

[3] の後ろなので、4

切り出してできた
adjectives リストの先頭

辞書？

- 配列の「何番目」ではないバージョンのようなもの。
  - 整数のほかに、実数や文字列が使える！
  - リストの「何番目」のことを、「インデックス」という。辞書では「キー」という。
- 波カッコ {} で囲み、「キーと値のペア」をカンマ区切りで入れる。
- *<キー>*:*<値>* でペア１個。キーと値はコロン：で区切る。

```
colors = {
       ‘red’ : ‘赤’,
       ‘green’ : ‘緑’,
       ‘blue’ : ‘青’}
```

※見やすくするため
改行を入れてあります。
１行でも書けます。

辞書の使い方

- キーに対応する値を見に行くには……
  - *<辞書名>*[*<キー>*]
  - 代入時以外で存在しないキーを使おうとするとエラー

```
print(colors[‘blue’])⏎
青
```

```
print(colors[‘pink’])⏎
KeyError: ‘pink’
```

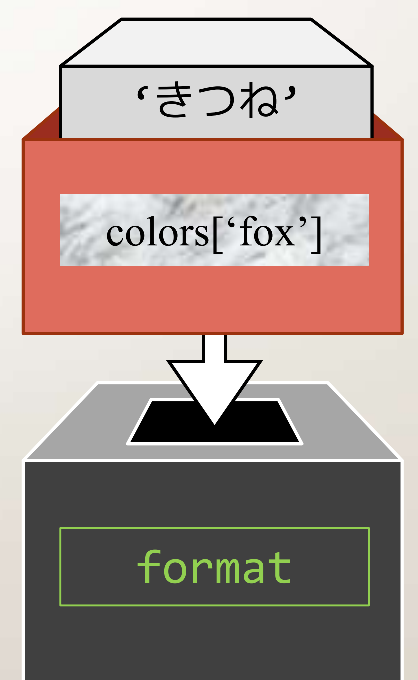
値の使い方

- やっぱり変数と全く同じ！
  - *<辞書名>* [*<キー名>*] をひとかたまりとして扱おう

後からキーを
登録することができる

```
colors[‘fox’] = ‘狐’⏎

‘こんがり{}色’.format(colors[‘fox’])
こんがり狐色
```



辞書の使い道

- やっぱり<*キー名*>は変数にできる
  - 存在しないキーを指定すると代入時以外はエラーなので注意！
  - リストよりも柔軟に使えるかも

```
name = input(‘何色?: ’) ⏎
print(colors[name])⏎
```

```
何色?: red ⏎
赤
```

ここまで OK ?

- リストというもの　……値の列
  - list = [1, 2.0, ‘three’]
  - アクセスするには list[0] 、先頭のインデックスはゼロ
  - 範囲外のインデックスを使おうとするとエラー
- 辞書というもの　……ペアの集合
  - dict = {‘red’:’赤’, ‘white’:’白’}
  - アクセスするには dict[‘red’] 、キー名を指定
  - 代入時以外で存在しないキーを使おうとするとエラー

演習 #1

- 次のように画面に表示するプログラムを、print 関数を用いて作成してください。
  ただし、print 関数の使用は 1 度だけです。

```
C: ’ちくわ大明神’
B: ’誰だ今の’
```

演習 #2

- ユーザーに 2 個の整数を入力させ、その値の和を表示するプログラムを作成してください。
ただし、下図のように「和は x です」という形で文字列とともに表示させてください。
- 余力があれば、実数の入力に対応してみてください。
- 数値以外がひねくれて入力されるケースは考えなくてかまいません。

```
1個目の値は? 123
2個目の値は? 432
和は 555 です
```

演習 #3

1. 自分の好きなものorこと 4 個を格納したリスト favorites を作成してください。
2. ユーザーに 1 から 4 までの整数を入力してもらい、それに応じて favorites 内の要素を左下のように表示するようにしてください。
3. 余力があれば、右下のような形で表示させてみてください。

```
何番目?（1-4): 4
英語
```

```
何番目?（1-4): 1
狐が1番目に好き
```

ヒント

- 演習 #1
  - 改行は「ある文字を入力する」or「文字列の囲み方を変える」と反映されます。
  - クォーテーション ‘ はそのままでは書けません。クォーテーションの直前に、ある文字が必要です。
- 演習 #2
  - 入力は input 関数で受け取れますが、返るのは文字列なのでそのままでは足し算できません。数値への変換が必要です。
  - {} のあるひな形をつくれば、あの関数で数値を埋め込めます。
- 演習 #3
  - インデックスは 0 から 3 の範囲なので、1 から 4 の入力に対応させるには、ちょっとだけ計算が必要です。
  - *<何番目>* というのは、変数に置き換えられます。