プログラミング実習I クラス5 (井村担当)

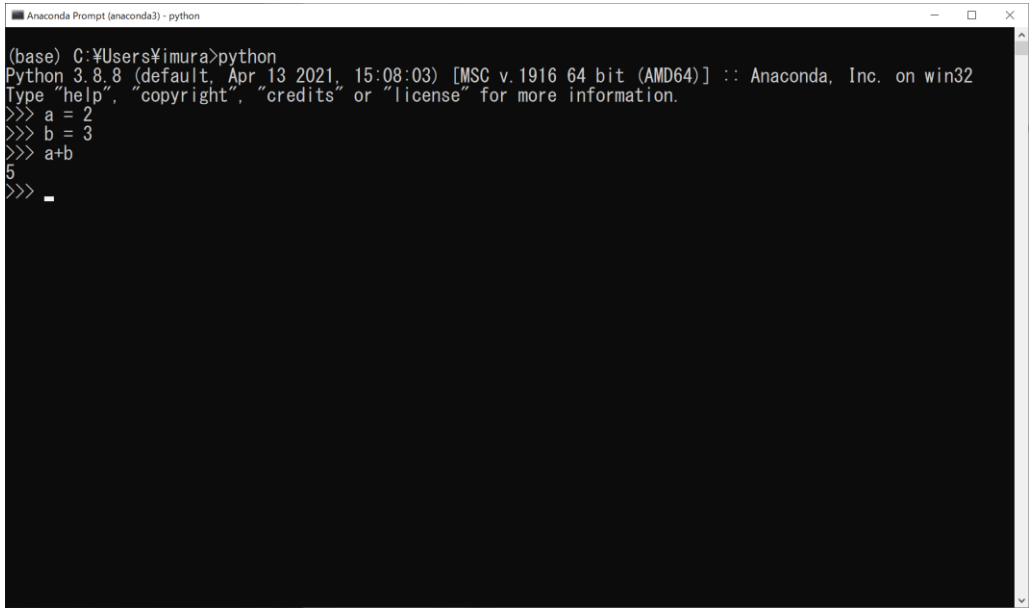
知能・機械工学課程 井村 誠孝

m.imura@kwansei.ac.jp

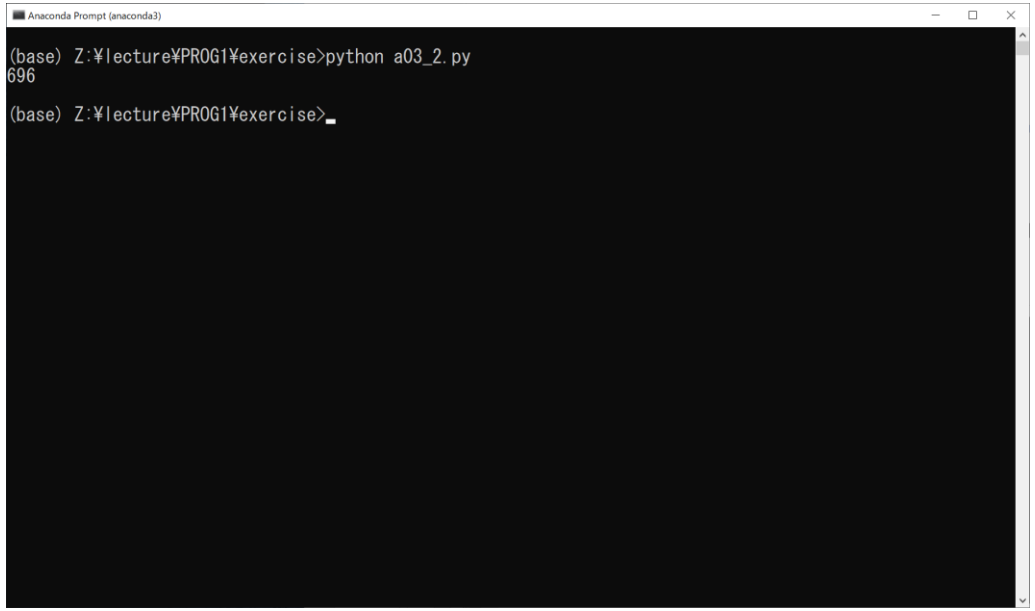
インタラクティブシェルとスクリプトの実行

Pythonの2つの実行方法

- インタラクティブシェル
  - Anaconda Prompt で
    python
    とのみ入力
  - Pythonの命令を1つずつ入力



- スクリプトを記述して実行
  - まずサクラエディタでスクリプト(ソースコード)を作成
  - Anaconda Prompt で
    python スクリプトファイル名
    と入力



インタラクティブシェルとスクリプトの実行の違い

- インタラクティブシェル
  - 1命令ずつキーボードから入力して実行する．
  - 1命令ごとに結果が返ってくる(何も表示されないこともある)．
  - 現在の状態には，シェルを起動してから今までの処理結果が反映されている．
  - 変数名を入力すると，その値が表示される．

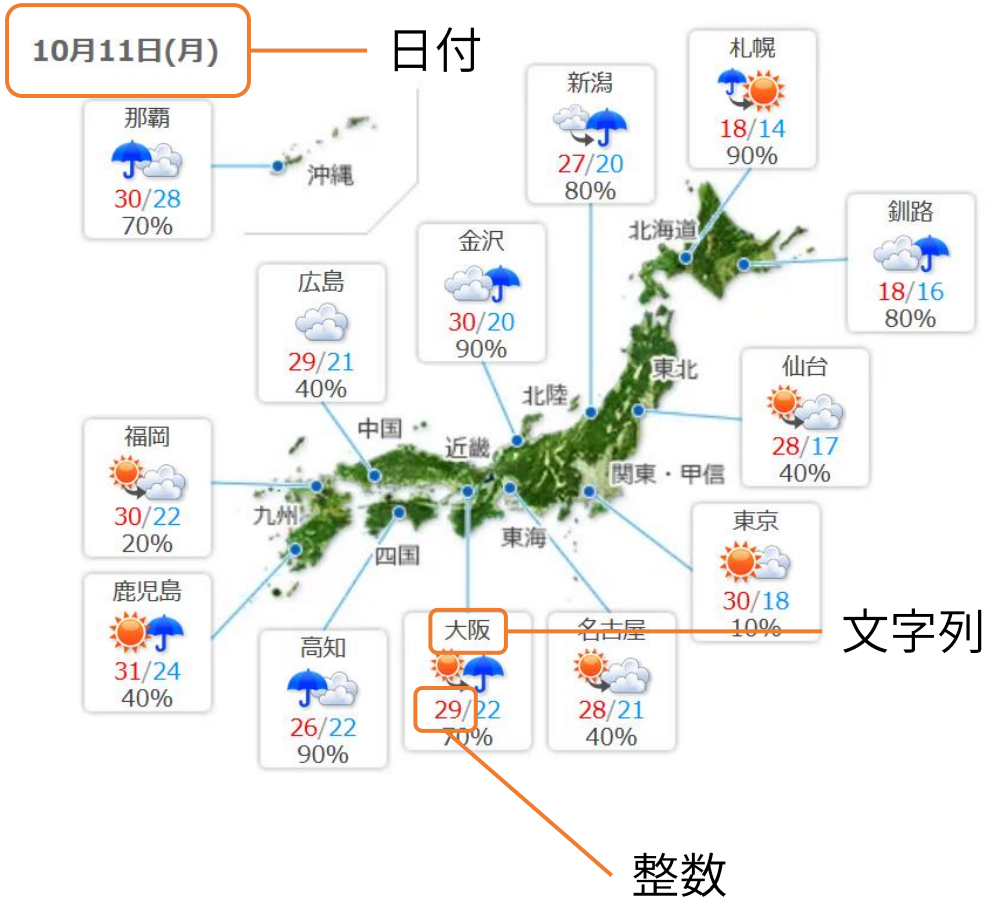
ちょっとしたテストに適している

- スクリプトの実行
  - スクリプト(ソースコード)という形で命令列を最初から最後まで記述する．
  - スクリプトを実行すると命令列が次々に実行される．
  - プログラムが終了するまで実行が続けられる．
  - 変数の値を表示するためには，関数 `print()` を使う必要がある．

ある程度まとまった処理を行う
プログラムはスクリプトで記述する

データ型と変数

データには様々な種類がある



文字列

コード	銘柄名	市場		株価		前日比		出来高	PER	PBR	利回り
6335	東京機	東1		1,618	S	+300	+22.76%	349,800	45.6	1.90	—
4956	コニシ	東1		1,835		+175	+10.54%	196,200	12.3	1.00	2.18
5698	エンビプロ	東1		2,092		+188	+9.87%	382,800	15.8	2.38	1.58
4337	ぴあ	東1		4,080		+350	+9.38%	125,500	—	23.00	—
8918	ランド	東1		12		+1	+9.09%	38,803,100	5.7	2.67	0.83
4331	T&Gニーズ	東1		1,250		+98	+8.51%	318,600	324	1.55	—
6962	大真空	東1		4,125		+320	+8.41%	423,500	20.8	1.16	1.21
9468	カドカワ	東1		6,320		+460	+7.85%	764,100	49.4	3.18	0.79
6175	ネットマーケ	東1		523		+37	+7.61%	678,400	20.8	2.59	1.15
8214	AOKIHD	東1		718		+50	+7.49%	1,406,700	46.9	0.50	1.39
3843	フリービット	東1		1,366		+95	+7.47%	1,128,700	43.8	2.84	—
7187	ジェイリース	東1		2,107		+138	+7.01%	1,217,800	17.6	12.76	1.42
3926	オープンドア	東1		2,612		+170	+6.96%	357,600	—	12.88	—
5423	東京製鉄	東1		1,108		+72	+6.95%	1,110,500	6.9	0.98	1.44
7033	MSOL	東1		3,415		+220	+6.89%	251,700	96.9	28.70	—

整数　　小数

これらをコンピュータで扱うには?

データ型

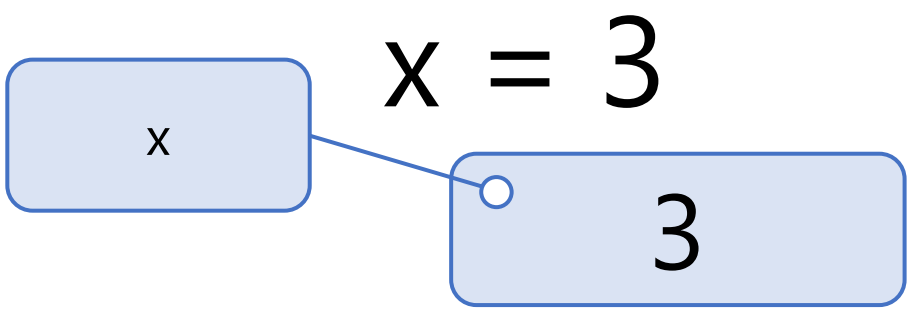
- データに応じたデータ型を利用する．
- 基本的なデータは組み込みデータ型で表現される．
  - 整数: `-1`, `0`, `256`
  - 文字列: `'Tokyo'`, `'東京'`, `'109'`
  - 小数: `-3.14`, `0.005`, `1.0`
  - 真偽: `True`, `False`
  - リスト: `[1,2,3,4,5]`, `['Tokyo', 'Osaka', 'Takamatsu']`

    他にもあります．詳しくは https://docs.python.org/ja/3/library/stdtypes.html 等を見てください．
- 組み込みデータ型は，データの書き方だけで型が区別される．

Pythonでは変数の型は自動的に判別されるが，
プログラムする際はデータ型を常に意識しよう

変数

- データ(数値，文字列，etc.)に付けられた名前



「x を 3 にする」
と考えておいてOK.

- 変数名は，アルファベット，数字，アンダースコア(_)で名付ける．
  - 先頭に数字を使うことはできない．
  - 先頭に _ を使うと，特別な意味を持つため，基本的には使用しない．
  - アルファベットの大文字小文字は区別される．
  - 言語仕様として特別な意味を持つ単語は使えない(`if`，`is`，`not`，`for`など)

変数名の制限

- 次のうち，変数の名前として使えないのは?

- ✗ `if`
- `when`
- ✗ `2days`
- `a12345`
- ✗ `hoge!hoge`
- `fuga_fuga`
- `ＡＢＣ` (注: 全角文字)
- ✗ `(*°-°)v`

予約語かどうかは，以下で確認できる．
https://docs.python.org/ja/3/reference/lexical_analysis.html#keywords

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

実はPython 3からは使用可能なのだが，
使用するべきでない．

型のいろいろ(1)

- 整数型: int型
  - 小数点を含まない数字
  - 扱える数の大きさに制限はない

わざわざ書いているのは，他の言語だと制限がある場合が一般的であるため．

- 小数型: float型
  - 小数点を含む数字
  - 精度に限界がある
    - 10進数ではきりがよい数字に見えても，コンピュータの内部(2進数)では無限小数になることが一般的である．

```
>>> 0.1 + 0.1 + 0.1
0.30000000000000004
>>> 0.2 + 0.2 + 0.2
0.6000000000000001
>>> 0.5 + 0.5 + 0.5
1.5
>>> 0.6 + 0.6 + 0.6
1.7999999999999998
>>>
```

0.5だけ誤差が生じないのは理由があります．

型のいろいろ(2)

- 文字列型: str型
  - シングルクォーテーション(') or ダブルクォーテーション(")で囲まれた文字列
    - 囲む記号はいずれを用いてもよい．
      - 'を含む文字列は"で囲む，といった使い方ができる．
      - 先頭と末尾では同じ記号を使う必要がある．
    - 'で囲まれた数値は文字列であり，数値として演算はできない．

```
>>> a = 'python'
>>> a
'python'
>>> type(a)
<class 'str'>
>>> a = "python"
>>> a
'python'
>>> type(a)
<class 'str'>
>>>
```

- 真偽型: bool型
  - 真(True)と偽(False)のいずれかの状態を表す
    - TrueとFalseは特別な意味を持つ予約語
  - 比較演算子による値の比較結果

```
>>> a = True
>>> a
True
>>> type(a)
<class 'bool'>
>>>
```

```
>>> a = 10
>>> a > 5
True
>>> a > 15
False
>>> a < 15
True
>>>
```

```
>>> a = 12
>>> b = 34
>>> a+b
46
>>>
>>> a = '12'
>>> b = '34'
>>> a+b
'1234'
>>>
```

型のいろいろ(3)

- リスト型: list型
  - 数値や文字列等を並べて格納するデータ型
  - 格納する要素を , (カンマ)で区切って並べ，全体を[]で囲む．
  - 変数名の後ろに[]を付けて番号を指定すると，要素が取り出せる．

```
>>> a = [1, 10, 100]
>>> a
[1, 10, 100]
>>> type(a)
<class 'list'>
>>> a[0]
1
>>> type(a[0])
<class 'int'>
>>>
```

  - 1つのリストの要素が異なるデータ型でもよい．

```
>>> a = ['睦月', 1]
>>> a[0]
'睦月'
>>> a[1]
1
>>>
```

詳しくは4章で

演算

計算ではなくて演算

- 演算は計算よりも広い概念
- 英語にすると operation
  - （機械などの）操作，運転
  - （手順・処置などの）実行，実施
- 演算方法を指示する記号列を演算子 operator と言う

演算子 operator

- 演算子はデータに作用して何らかの結果を返す
- 算術演算子
  - +, -, *, /: 和，差，積，商
  - // : 小数点以下を無視する除算
  - % : 剰余(余り)
  - ** : ベキ乗
- 比較演算子
  - <, <=, >, >=, !=, ==: 左辺と右辺を比較し，真偽型(True / False)を返す
- 代入演算子
  - =
  - +=, -=, *=,...: 複合代入演算子

代入演算子の正確な作用: 左辺に置かれた変数が，右辺に置かれたデータを参照するようになる．

演算子の優先順位

高

優先順位

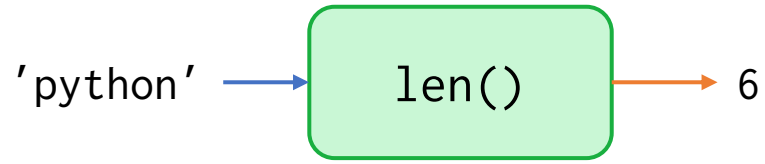
低

演算子	説明
(expressions...), [expressions...], {key: value...}, {expressions...}	結合式または括弧式、リスト表示、辞書表示、集合表示
x[index], x[index:index], x(arguments...), x.attribute	添字指定、スライス操作、呼び出し、属性参照
await x	Await 式
**	べき乗
+x, -x, ~x	正数、負数、ビット単位 NOT
*, @, /, //, %	乗算、行列乗算、除算、切り捨て除算、剰余
+, -	加算および減算
<<, >>	シフト演算
&	ビット単位 AND
^	ビット単位 XOR
|	ビット単位 OR
in, not in, is, is not, <, <=, >, >=, !=, ==	所属や同一性のテストを含む比較
not x	ブール演算 NOT
and	ブール演算 AND
or	ブール演算 OR
if -- else	条件式
lambda	ラムダ式
:=	代入式

関数とメソッド

関数

- 関数: ある特定の処理を行ってくれる命令のまとまり




```
n = len('python')
      引数
```

戻り値(あるいは返値): この例では 6 になる

- 2.3節で紹介されている例
  - `len()`: 長さを返す
  - `str()`: 文字列に変換する
  - `print()`: 画面に出力する
  - `range()`: 整数の並びを作る
  - `list()`: リストに変換する

あらかじめ用意されている関数(組み込み関数)

組み込み関数

A abs() aiter() all() anext() any() ascii() **B** bin() bool() breakpoint() bytearray() bytes() **C** callable() chr() classmethod() compile() complex() **D** delattr() dict() dir() divmod()	**E** enumerate() eval() exec() **F** filter() float() format() frozenset() **G** getattr() globals() **H** hasattr() hash() help() hex() **I** id() input() int() isinstance() issubclass() iter()	**L** len() list() locals() **M** map() max() memoryview() min() **N** next() **O** object() oct() open() ord() **P** pow() print() property()	**R** range() repr() reversed() round() **S** set() setattr() slice() sorted() staticmethod() str() sum() super() **T** tuple() type() **V** vars() **Z** zip() **_** __import__()

https://docs.python.org/ja/3/library/functions.html

メソッド

- 特定のデータ型に結び付けられた関数
- 2.4節で紹介されている例
  - 文字列型のメソッド
    - split(): 文字列を分割してリストを返す
    - upper(): 大文字に変換した文字列を返す
- ここで紹介されているメソッドはごく一部
  - 各データ型に対して，メソッドが通常は複数存在する
  - 様々な機能を提供するデータ型が存在する

```
address = 'Tokyo, Japan'
print(address.split(','))
```

文字列型の変数 address のメソッド split

全部を覚える必要は全くありません(無理)