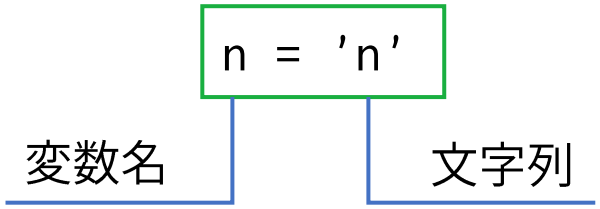
プログラミング実習I クラス5 (井村担当)

知能・機械工学課程 井村 誠孝

m.imura@kwansei.ac.jp

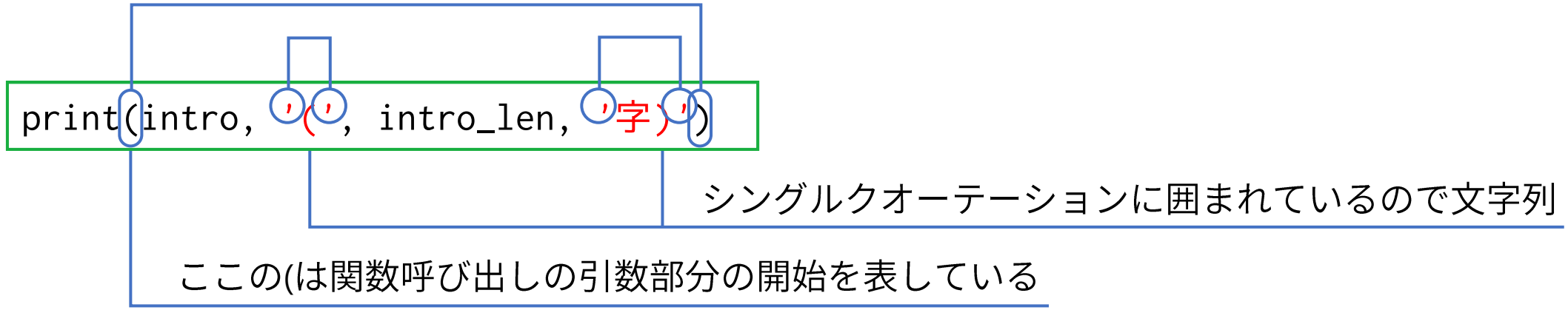
前回補足

- 文字列と変数名を区別しよう
  - シングルクオーテーション ' で前後を囲まれていたら文字列
    - ダブルクオーテーション " の場合もある
    - カンマ , は似ているが上下位置が違うので異なる記号



人は前後の文脈で判断できることでも，
コンピュータは判断できない
→人が形式を整える必要がある

- 対で現れる記号を確認すると，構造がわかる



前回補足

- 足せる? 足せない?

```
full_name = 'Nobita' + ' ' + 'Nobi'
```

文字列の連結 → できる

```
given_name = 'Nobita'
family_name = 'Nobi'
full_name = given_name + ' ' + family_name
```

変数と文字列の連結
→ 変数の値が文字列型
→ できる

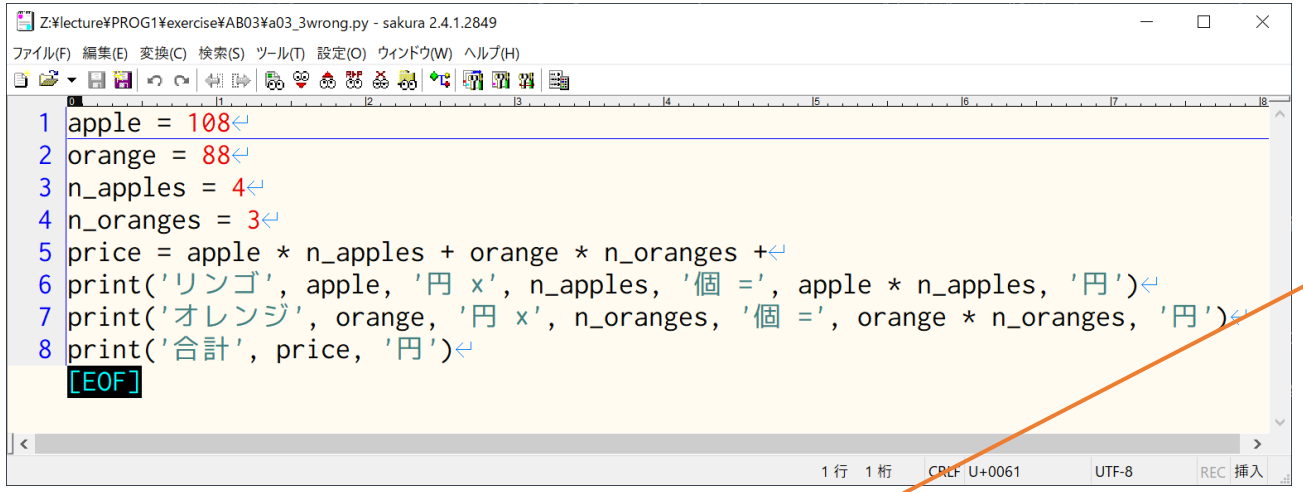
```
given_name = 'Nobita'
age = 10
name_and_age = given_name + ' ' + age + ' years old'
```

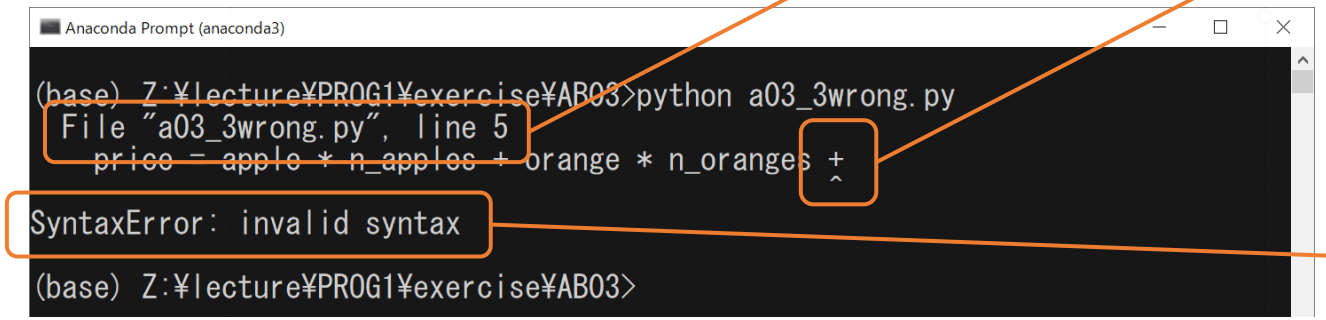
変数と文字列の連結
→ 変数の値が整数型
→ できない

- 変数は「値が更新される」場合と「値が使用される」場合がある.
- 変数を使用する(値が使用される)場合は，値の型が何であるかに気を配ろう.

前回補足: エラーメッセージの読み方

- エラーメッセージを確認すると問題が絞り込める.





5行目に問題があることがわかる

+ の下に ^ が表示されている
→この付近に問題がありそうなことがわかる.
ただし，もっと前に問題があり，ここでようやくわかったという場合もあるので，前も探す.

エラーの内容: SyntaxError: invalid syntax
英語だが翻訳してみる.
「構文エラー: 無効な構文」
構文に誤りがある.
エラーメッセージで検索してみるのも有益.

変数の値を画面に出力する推奨方法: f文字列

- 関数print()の引数に，カンマ(,)で区切って並べるの，面倒だ…

- そんなあなたに「f文字列」
  - 文字列の直前に f を付ける．
  - 変数の値を出力したい場合は，文字列中に直接，{}で囲んだ変数名を記述する．
- 例

f文字列がないとき…

```
print('合計', price, '円')
```

f文字列があるときー!!

```
print(f'合計 {price} 円')
```

```
print(intro, '(', intro_len, '字)')
```

```
print(f'{intro} ({intro_len}字)')
```

{}を使うと，どこが変数なのか見分けがつきやすい

データ型

- 型 (2.2節)
  - プログラムに登場するデータには，全て型がある．
  - 基本的な型は組み込み型と呼ばれる．
    - ソースコードでのデータの表現方法に基づいて，型が自動的に決められる．

ソースコード: 人が理解できるプログラムの表現．
Pythonではスクリプトと同じ

種類	型名	表記法	例	備考
整数	int	整数を小数点なしで書く．	`-1 0 256`	
小数	float	小数点を付けて数字を書く．	`-3.14 .005 1.0`	科学的表記法も可能 例: 1e-3 (=1000)
文字列	str	引用符(‘または”)で囲む．	`'Tokyo' '東京' "109"`	複数行にわたる文字列を”’…”’で記述可能
真偽	bool	True か False の予約語を書く．	`True False`	文字列ではないので”や””で囲まない
リスト	list	カンマで区切ってデータを並べ，[]で囲む．	`[1, 2, 3, 4, 5]` `['Tokyo', 'Osaka', 'Takamatsu']`	空のリストは []

標準ライブラリで確認しよう

- 組み込み型の一覧

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

- 組み込み型
  - 真理値判定
  - ブール演算 --- and, or, not
  - 比較
  - 数値型 int, float, complex
  - イテレータ型
  - シーケンス型 --- list, tuple, range
  - テキストシーケンス型 --- str
  - バイナリシーケンス型 --- bytes, bytearray, memoryview
  - set（集合）型 --- set, frozenset
  - マッピング型 --- dict
  - コンテキストマネージャ型
  - Type Annotation Types --- Generic Alias, Union
  - その他の組み込み型
  - 特殊属性

様々なデータ型

- 組み込み型を利用することで様々なデータを表現可能
  - 例: 日付を表現したい

```
birthday_year = 2112
birthday_month = 9
birthday_day = 3
```

日付が登場するたびに変数3つ作って
管理するのは面倒だな…

- 特定の目的のための専用のデータ型があると便利
  - 例: 日付を表すためのdate型

```
import datetime
birthday = datetime.date(2112, 9, 3)
```

様々なデータ型は
モジュールとして提供されており
利用する際は import が必要

Pythonには便利なデータ型が多数用意されている

標準ライブラリで確認しよう・その2

- 標準モジュールの一覧

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

非常に多数用意されている

- テキスト処理サービス
  - string --- 一般的な文字列操作
  - re --- 正規表現操作
  - difflib --- 差分の計算を助ける
  - textwrap --- テキストの折り返しと詰め込み
  - unicodedata --- Unicode データベース
  - stringprep --- インターネットのための文字列調製
  - readline --- GNU readline のインターフェース
  - rlcompleter --- GNU readline向け補完関数
- バイナリデータ処理
  - struct --- バイト列をパックされたバイナリデータとして解釈する
  - codecs --- codec レジストリと基底クラス
- データ型
  - datetime --- 基本的な日付型および時間型
  - zoneinfo --- IANA time zone support
  - calendar --- 一般的なカレンダーに関する関数群

メソッド

- 特定のデータ型と機能とはセットになっていることが一般的
  - 文字列 → 分割する，大文字にする，etc.
  - 日付 → 曜日を調べる，文字列に変換する，etc.
- 機能を使用する=メソッドを実行する
  - ピリオド(.)を使うことで，どの変数(あるいはデータ型)のメソッドであるか，を指定する(3.3節)
- データ型とメソッドの関係については，3.2節〜3.4節の例を用いた説明を参照
  - メソッドの分類 (pp. 72-74)
    - 内容を変えずに結果を返すメソッド
    - 内容を変えて結果は返さないメソッド
    - 実体が無くても使えるメソッド

datetimeモジュールの使用例

- datetimeモジュールは日付や時刻を操作するためのデータ型を提供している．
- モジュールを使う際には import 文を必要とする．

```
import datetime
```

- モジュールの中に各種の型が定義されている．
- 型と同じ名前のメソッド(初期化メソッド)でデータ型の実体(インスタンス)を適宜生成して使用する．
  - モジュールAで提供されているデータ型Bを使用する場合は，A.Bと記述する．

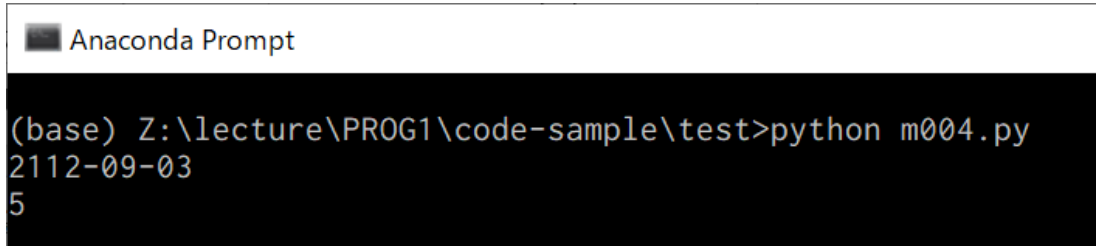
```
day = datetime.date(2112, 9, 3)
```

date型のメソッドを使う

- date型のメソッドの例: weekday()

```
import datetime

day = datetime.date(2112, 9, 3)
print(day)
print(day.weekday())
```



- 実体がなくても使えるメソッドの例: today()

```
import datetime

day = datetime.date.today()
print(day)
```

datetimeモジュールの他の型

- モジュールには関連する複数の型が用意されている場合がある．
- datetime型の使用例
  - インスタンスの生成

```
import datetime

day = datetime.datetime(1969, 7, 21, 5, 17, 40)
print(day)
```

datetime.datetime とdatetimeが2回登場するが，
前はモジュール名，後ろはデータ型名である．
ややこしいが仕方ない．

Anaconda Prompt

```
(base) Z:\lecture\PROG1\code-sample\test>python m006.py
1969-07-21 05:17:40
```

  - 今の日時のインスタンスを生成

```
import datetime

day = datetime.datetime.now()
print(day)
```

Anaconda Prompt

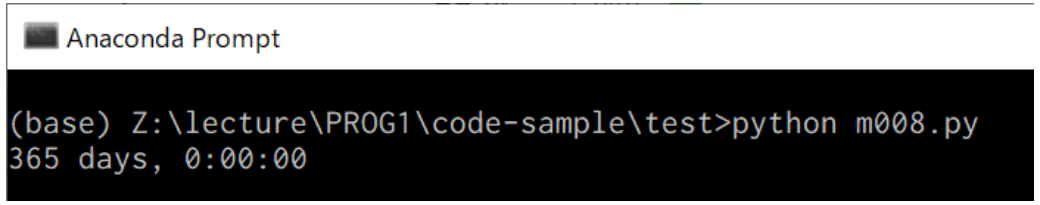
```
(base) Z:\lecture\PROG1\code-sample\test>python m007.py
2022-10-14 04:13:19.480621
```

date型を用いた計算

- date型間で減算を行うと，経過日数が計算される．

```
import datetime

first_day_2022 = datetime.date(2022, 1, 1)
first_day_2023 = datetime.date(2023, 1, 1)
days_2022 = first_day_2023 - first_day_2022
print(days_2022)
```



- 減算の演算子 - がdate型に適用可能なのは，date型が演算子をオーバーロード(operator overloading; 上書き定義)しているため．
- 試してみよう
  - date型間の足し算は可能か?
  - 過去から未来を引くとどうなるか?

date型の減算により得られるデータの型は?

- インタラクティブシェルで確認してみよう.

```
>>> import datetime
>>> a = datetime.date.today()
>>> b = datetime.date(2112, 9, 3)
>>> d = b - a
>>> type(d)
<class 'datetime.timedelta'>
```

datetimeモジュールで提供される
timedelta型であることがわかる

- help()で関連する情報を得ることができる.

```
>>> help(d)
Help on timedelta object:

class timedelta(builtins.object)
 |  Difference between two datetime values.
 |
 |  timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)
 |
 |  All arguments are optional and default to 0.
 |  Arguments may be integers or floats, and may be positive or negative.
```

ソースコードに登場するものを区別しよう

- 値(数値，文字列，...)なのか，変数(や関数，メソッド)名なのか
  - 数字で始まる = 値(数値)
  - 前後に引用符がある = 値(文字列)
  - True と False (引用符なし) = 値(真偽値)
- 変数なのか，関数なのか
  - 名前の後ろに()が付いていれば，関数あるいはメソッド
    - ()の中に引数がある場合も，無い場合もある．
- モジュール名なのか，データ型なのか
  - データ型はモジュール名より右に現れる．
    - モジュールの中でデータ型が定義されているため．
      - ピリオド(.)は，左のものが右のものを所有していることを表している．口語的には「の」，文語的には「が所有している」と読めば理解しやすい．

例

```
import datetime

day = datetime.datetime.now()
print(day)
```

2つのdatetimeの違いに注目

```
import datetime

day = datetime.date.today()
print(day)
print(day.weekday())
```

```
import datetime

first_day_2022 = datetime.date(2022, 1, 1)
first_day_2023 = datetime.date(2023, 1, 1)
days_2022 = first_day_2023 - first_day_2022
print(days_2022)
```

dateは型名だが，後ろに()が付いているので
ここでは初期化メソッド

値(数値)
モジュール名
データ型名
メソッド名
変数名
組み込み関数
予約語