2. 簡単な計算：Getting started through simple calculations

プログラミング・データサイエンス I

2022/4/20

1　今日の目標

今日の目標

- データの表現について知る
- プログラミング環境に慣れる
- プログラムで使う基本演算を知る

コンピュータが扱うデータについて、その概要を知ることから始めましょう。また、プログラミング環境に慣れることが一番の目標にして、実際にプログラムを少し書き、実行しましょう。

プログラミング言語は、「言語」ですから、始めは、わからないことが出てきても、いちいち気にする必要はありません。次第に慣れていきます。最初に、基本的な演算 (計算) を行っていきます。それ以外のことも、少し出てきますが、「だいたい」のイメージがつかめれば十分です。

1.1　コンピュータとデータ

私達が使っているコンピュータが、数字や文字などのデータをどのように扱っているを理解することから始めましょう。例えば、インターネットを通じて、Web を見ている場合を考えましょう。Web のページの中には、テキストや画像があります。また、動画や音声のリンクがあるかもしれません。これらは、どのような形式になってインターネットを流れ、コンピュータで再生しているのでしょうか。

コンピュータとインターネットでは、全てのデータ、つまり数値、文字、画像、音声、動画などを、2 進数で表現しています。私達が日常生活で使っている数字は 10 進数と言

います。0 から 9 までの 10 種類の記号を用い、$9+1=10$ と桁上りする仕組みが 10 進数です。一方、2 進数とは、0 と 1 の二種類の記号を用い、$1+1=10$ と桁上りする仕組みのことです。記号の種類が 2 つしかないことと、演算規則が簡単であることから、2 進数は電子回路で容易に実装可能です。このことが、コンピュータとインターネットが 2 進数を使っている理由です。2 進数一桁を bit と言います。

文字には、対応する数値が定まっています。数字やアルファベットなどは、2 進数 8 桁 (1Byte という) で表現しています。先頭の bit を 0 に固定した部分を ASCII コードと言います。2 進数 7 桁は 10 進数の 0 から 127 までの数字を表すことができますから、ASCII コードは 128 種類の文字を表現できます。

日本人は、日常的に数千文字を使うと言われています。1Byte では、かなや漢字を表現できません。そこで、かなや漢字は 2Byte で表現します。現在では、世界中の様々な文字に対してコードが定義されています。UNICODE というコード体系を使って、様々な言語の文字にコードが定義されていることがわかります。この中には、漢文の返点、チェスの駒の文字、さらに絵文字なども定義されています。[*1]

音声や画像は、もともとはアナログ、つまり連続的なデータです。こうした連続的データをコンピュータで処理する際には、サンプリングという離散化、つまり飛び飛びの値に変換しています。コンピュータは、これらの離散的値を、さらに 2 進数表現に変換しています。

1.2 プログラムとデータ

プログラムは、データを加工し、結果をデータとして出力します。このときのデータとは、どのようなものでしょうか。

データには、数字や文字列のような単純なものだけではありません。例えば、画像のデータを考えましょう。画像のデータは、点の集まりです。コンピュータの画面もデジタル画面で撮影した画像も、点の集まりです。これらの、各点には色を設定します。点の総数、画像の縦横比、各点の色の表現方法等が組になってデータとなります。つまり、単純な値の羅列ではなく、取り扱い方法を定める構造があります。

もちろん、プログラムでは、構造を持ったデータを扱うことができます。例えば、データが列になったリスト、鍵と値の組になったものなどです。前述のような画像も対応した構造とともに、扱います。詳しくは 5 回目以降で扱います。

[*1] `http://www.unicode.org/charts/`

課題 1.1　自分の PC 中にある、テキストファイル、Word ファイル、画像ファイルについて、その大きさを確認しなさい。ファイルの大きさの単位に注意しなさい。

2　プログラムの書き方

2.1　サンプルプログラムの取得

サンプルプログラムの取得

- Github からサンプルプログラムを取得
- ダウンロードしたファイルを確認

始めに、今回使用するサンプルプログラムをダウンロードしましょう。作業用フォルダにマウスを合わせて右クリックし、VSCode を起動します。

左下の歯車のアイコンを押し、一番上の「コマンドパレット」を選びます。現れたサブウィンドウの検索窓に「Git」と入れると、「Git クローン」があります。それを選ぶと、リポジトリ名を聞いてきますから、以下を入力します。

`https://github.com/first-programming-saga/fundamentals`

保存先を聞いてきますから、作業用フォルダ `PythonProject` を指定します。作業用フォルダ `PythonProject` の下に、`fundamentals` というフォルダが出来ています。その下に、ファイルが出来ました。

もしも、「Git クローン」がない場合には、以下のようにしてサンプルプログラムを取得します。

- 作業用フォルダにマウスを合わせて右クリックし、ターミナルを開きます。
- ターミナル上で以下のコマンドを実行します。

```
git clone https://github.com/first-programming-saga/fundamentals
```

- コマンドを実行したフォルダに、`fundamentals` というフォルダが出来て、ファイルが保存されます。

2.2 プログラムの書き方: 1

プログラムの書き方: 1

- フォルダの開き方
- Cell の中にプログラムを書く
  - Cell 毎に実行できる
  - 全てのセルを上から順に実行することもできる
- Cell Type を Markdown とすると、テキストとなり、説明に使える
- プログラム中のコメントは”#”で開始

`fundamentals` フォルダにマウスを合わせて、右ボタンを使って VSCode を開きます。それでは、`fundamentals` フォルダ中のプログラム `firstPythonProgram.ipynb` を開きましょう。

今回サンプルプログラムでは、セルの外側に日本語で説明が書かれている部分があります。このように、セルのタイプを”Markdown”と設定することで、説明を追加することができます。このテキスト部分は、プログラムとしては実行しません。

プログラム中に”#”を書くと、そこから行末までがコメントです。つまりプログラムとして実行しない部分です。

2.3 プログラムの書き方: 2

プログラムの書き方: 2

- 文の区切り
  - 改行か”;”(セミコロン)
- 行途中での折り返し
  - バックスラッシュ”\”を使って、継続していることを示す。
    ”¥”でも同じ
- 大文字と小文字は区別
- 変数、演算子、関数の間のスペースは見やすいように適当に入れてよい
- 行の先頭のスペースは意味がある
- 予約語がある

プログラムの書式をまとめておきます。後で、例を見ながら確認すれば十分です。

一つの文は、改行で区切るのが基本です。どうしても、一行に複数の文を書きたい場合には、セミコロンで区切ります。

一行が長くなってしまう場合には、適当なところで折り返すと見やすくなります。その場合には、バックスラッシュ”\”を使います。日本語キーボードでは、円マーク”¥”です。ただし、後で見るように、関数の引数並びや、データ定義では、複数行に分けて書くことができる場合があります。

プログラムを書く上で重要なことの一つに、大文字と小文字の区別があります。別の文字として扱われます。注意してください。

変数、演算子、関数の間には、見やすいように適当にスペースを入れます。ただし、行の先頭のスペースには意味があります。5 回目以降に説明します。

プログラミング言語としてあらかじめ定義されている単語 (予約語) があります。

3 計算と代入

計算と代入

- 基本的な演算記号
- イコール記号の意味

3.1 基本的演算

演算子	例	説明
+	a + b	加算 $a+b$
-	a - b	減算 $a-b$
*	a * b	乗算 $a \times b$
/	a / b	徐算 a/b
//	a // b	a を b で除した整数部分 $\lfloor a/b \rfloor$
%	a % b	a を b で除した余り $a \bmod b$
**	a ** b	a を b 回乗ずる a^b

表 1　基本的演算

最初に演算記号を見ていきます (表 1)。四則演算の記号は大丈夫でしょうか。掛け算が*となっています。キーボードに「×」が無いですからね。

ソースコード 3.1　firstPythonProgram.ipynb の最初のセル

```
n = 10 #n という変数に値 10 を代入　int(整数) 型
m = 3 #m も整数型
a = n + m
b = n * m
c = n / m
d = n % m  #n を m で除した余り
e = n ** m #n の m 乗
print(a, b, c, d, e)
print(type(n)) #型を確かめる
```

表 1 中にある最後の三つは少し説明が必要ですね。a//b は、a を b で割ったときの整数部分を返します。例えば、1//0.3 を実行すると 3.0 となります。a%b は、a を b で割った時の余りを返します。1%0.3 は、0.1 を返します。本当のことを言うと、ちょうど 0.1 とはなりません。a**b は、a の b 乗です。数学の式で表すときは、a^b です。b は整数である必要はありません。

1%0.3 がちょうど 0.1 にならない理由です。コンピュータの中は、2 進数で動いてることを説明しました。整数は良いのですが、問題は小数です。1/2 や 1/4 のような、2 のべき乗の逆数は正確に表現できます。しかし、0.3 は 2 のべき乗を使って有限桁では正確に表現できないため、有効桁の一番最後にちょっと誤差が入っています。これが理由です。

3.2　代入

もっとも注意が必要な記号は、=です。これは、両辺が「等しい」という記号ではありません。この記号は、右辺を計算した結果を、左辺に代入することを表しています。「等しい」という記号は、後で出てきます。

それでは、firstPythonProgram.ipynb を見ていきましょう。ソースコード 3.1 は、簡単な計算をする部分です。対象となるセルを実行しましょう。見ただけで、何をするかは概ねわかりますね。そのセルだけを実行して、確かめましょう。

4 変数と型

4.1 変数と型

数値の演算: calculating values

- 数値は、桁の制限があることに注意
- 数型: int 型
- 浮動小数型: float 型
- 複素数型: complex 型
  - 虚数単位は `j`
  - 例: `a = 3 + 2j`

コンピュータの中では、全てのデータを 2 進数で表現します。データは、整数であったり、小数であったり、文字であったりします。プログラムでは、それらをどのように区別しているのでしょうか。

前節で見たように、データは変数に保存します。ソースコード 3.1 中で、=の左側にある、つまり右側の値を代入する先が変数 (variables) です。Python では、全ての変数に型があり、これでデータの種類を区別しています。ソースコード 3.1 の最後の行で、変数 `n` が `int` であること、つまり整数型であることを確認しています。`type()` は、括弧の中に入るモノの型を調べる関数です。関数については、後で説明します。

次に、二番目のセル (ソースコード 4.1) を見てください。最初の行で `x = 10.` と書いています。これは、変数 `x` に `10.0` という数値を代入しています。つまり `x` は整数ではありません。プログラミング言語では、このように小数点を含む型を「浮動小数点型 (floating point type)」と呼びます。「浮動小数」とついているのは、各桁の数値と小数点の位置という浮動する情報を保持しているからです。なお、プログラムでは、小数点以下が 0 の場合には、小数点以下を省略することができます。

課題 4.1　ソースコード 4.1 において、`x` と `y` に別の値を設定し、その動作を確認しなさい。

変数に型があるので、相互に変換する必要が生じます。ソースコード 4.2 では、整数型と浮動小数点型を相互に変換しています。浮動小数点型から整数へ変換すると、小数点以下を切り捨てます。

ソースコード 4.1 =

firstPythonProgram.ipynb の二番目のセル

```
x = 10. #x という変数に値 10.0 を代入 float(浮動小数) 型
y = 3. #y も浮動小数型
a = x + y
b = x * y
c = x / y
d = x % y  #n を m で除した余り
e = x ** y
print(a, b, c, d, e)
print(type(x))
```

ソースコード 4.2 =

firstPythonProgram.ipynb の三番目のセル

```
n = 10
x = float(n) # 浮動小数への変換
print(type(n), type(x))
m = int(x) # 整数への変換
print(type(m))
print(n, x, m)
```

4.2 文字列型とその演算

文字列型とその演算

- 文字列の代入
- 文字列と数値の変換

プログラミングの対象として、文字列も重要です。文字列データの処理、出力文字列の生成など、様々なところで必要となります。

Python では、文字列はシングルクォーテーションまたはダブルクオーテーションで表します。二つのクオーテーションに役割の区別はありません。ただし、開始と終了は同じ記号でなければなりません。

また、シングルクォーテーションを含む文字列は、ダブルクオーテーションで表します。その逆に、ダブルクォーテーションを含む文字列は、シングルクォーテーションで表

ソースコード 4.3　firstPythonProgram.ipynb の四番目のセル

```
s = 'string'
t = '"string"'
u = "string"
print(s, t, u)
```

ソースコード 4.4　firstPythonProgram.ipynb の五番目のセル

```
s = 'saga'
u = "university"
t = s + ' ' + u
print(t)
```

します。ソースコード 4.3 を見てください。t という変数には、ダブルクォーテーションで囲まれた文字列が保存されます。実行結果を確かめてください。

文字列の連結は + の記号を使います。ソースコード 4.4 を見てください。3 行目で三つの文字列を連結しています。

4.3　文字列と数値

文字列と数値

- 数値を文字列に変換
- 文字列を数値に変換
- 文字列と数値が混じった文字列

数値と文字列の間の変換を考えましょう。つまり、ある数字を表現する文字列、文字列としての数字の列に対応した整数や浮動小数です。プログラム firstPythonProgram.ipynb の 6 番目と 7 番目のセルです。

出力時に、文字列と数値を連結してプリントしたい場合があります。しかし、文字列と数値は型が異なるので、+ で結ぶことができません。数値を文字列化し、+ で連結するのが一つの方法です。もう一つは、文字列の前に f というキーワードをつける方法があります。f はフォーマット、つまり書式を定めるという意味です。{}の中に、変数名を指定します。

ソースコード 4.5　数値と文字列の相互変換

```
x = 1.25
xs = str(x)
y = float(xs)

print(type(x), type(xs), type(y))
print(x, xs, y)
```

ソースコード 4.6　数値と文字列の相互変換

```
x = 125
xs = str(x)
y = int(xs)
print(type(x), type(xs), type(y))
print(x, xs, y)
```

ソースコード 4.7　数値と文字列の相互変換

```
x = 1.45
s = '答えは'+str(x)
print(s)
s = f'答えは{x}'
print(s)
```

4.4　部分文字列

部分文字列

- 先頭は 0
- 最末尾は-1
- :を使って範囲を指定

文字列変数は、その一部分を切り出すことが可能です。文字列には位置の番号があり、先頭が 0 です。最後尾からも数えることができます。末尾が-1 です。範囲を指定することもできます。

ソースコード 4.8　文字列の位置

```
alphabet = "abcdefghijklmnopqrstuvwxyz"
print(alphabet[5])
print(alphabet[-10])
print(alphabet[0:5])
print(alphabet[-10:-1])
```

5　複合代入演算子

5.1　複合代入演算子

複合代入演算子

- ある変数に演算を行い、元の変数に代入
- 慣れると、プログラムが書きやすい
  - 慣れるまでは、無理に使わない

演算子	例	説明
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
//=	a //= b	a = a // b
%=	a %= b	a = a % b
**=	a **= b	a = a ** b

表 2　複合代入演算子

Python には、プログラムを効率的に書くための複合代入演算子があります (表 2)。これらは、他のプログラミング言語にもよく見られます。

一つだけ説明します。最初の `a += b` を見てください。これは `a = a + b` と同じ意味です。`a = a + b` だと、両辺に `a` が現れて、=の記号が気持ち悪いと感じる人も居るでしょう。`a += b` と書くと、変数 `a` に `b` を足すという意味が、慣れると、わかりやすくなります。また、`a += b` のほうが少しだけ短くなります。ソースコード 5.1 に簡単な例を

ソースコード 5.1　複合代入演算子の例

```
s = 0
s += 1 # s = s + 1 と同じ
s += 2
s += 3
print(s)
```

ソースコード 5.2　複合代入演算子の例

```
s3 = 0
for k in range(1,11):#k を 1 から 10 まで変化させる
    s3 += k
print(s3)
```

示します。

複合代入演算子は、慣れると非常に便利です。慣れない場合には、使う必要はありません。中途半端な理解で使うのは、むしろ危険です。

5.2　compoundOperators.ipynb

compoundOperators.ipynb

- 複合代入演算子の使用例
- データの和を計算する

1 から 10 までの和計算もしてみましょう (ソースコード 5.2)。3 番目のセルでは、s3 という変数に 1 から 10 まで足しています。for のところは、まだ扱っていませんが、k の値が 1 から 11 の一つ手前まで変化しながら繰り返します。繰り返しについては、また後日説明します。

6 論理演算: Logical operations

6.1 論理演算: Logical operations

論理演算: Logical operations

- 二つの論理値
  True, False
- 論理演算
  and, or, not
- 比較演算の結果は論理値になることに注意
  論理演算可能

演算子	例	説明
==	a == b	a と b の値が等しい
!=	a != b	a と b の値が等しくない
>	a > b	a の値は b の値より大きい
>=	a >= b	a の値は b の値以上
<	a < b	a の値は b の値より小さい
<=	a <= b	a の値は b の値以下
is	a is b	a と b は同じオブジェクト
is not	a is not b	a と b は異なるオブジェクト

表 3　比較演算

最初の回に、「条件分岐」というキーワードが出てきました。ある条件を満たすとき、満たさないときに、異なる処理をするものです。

「条件」を記述するためには、比較という操作が必要になります。「等しい」、「以上」などです。表 3 を見てください。== が「値が等しい」ということを判定する記号です。

「オブジェクト」は馴染みがないかも知れません。オブジェクト (object) とは、モノという意味があります。これまで見てきた変数では、整数や浮動小数は数という一つの値でした。しかし、文字列は、文字の集まりですね。この後、さまざまな値の塊がでてきます。python では、変数は全てオブジェクトとして扱います。「同じオブジェクト」という

ソースコード 6.1　論理演算とその結果の値

```
#Bool 型のテスト
x = 8
a1 = (0 <= x < 10)
print(a1)
a2 = (x >= 10)
print(a2)
a3 = ((0 <= x) and (x < 10))
print(a3)
a4 = (not a2)
print(a4)
```

ソースコード 6.2　論理演算とその結果の値

```
p = True
q = False
r = (p and q)
print(r)
```

のは、わかりにくいので、少し先で説明します。

比較を行うと結果は、「真 (True)」と「偽 (False)」のいずれかになります。True か False の値をとる変数のことを Boolean (ブール型) と言います。論理型なので、論理演算できます。and は「かつ」、or は「または」、そして not は否定です。

booleanTest.ipynb を開きましょう。最初のセルは、比較した結果がどのような値となっているかを調べています (ソースコード 6.1)。3 行目の右辺を見てください。() で論理演算を囲うことで、その結果を左辺に代入しています。実行すると、それぞれの結果が True または False となって返ってきていることがわかります。

二番目のセル (ソースコード 6.2) では、二つの論理変数 p と q に値を代入しています。その後で、論理積を r に代入しています。

6.2　オブジェクトの比較

三番目のセルを見てください (ソースコード 6.3)。abc という文字列を str0 に代入しています。str1 に str0 を代入したので、str1 にも abc が入っています。もっと大事なことは、str0 と str1 は、どこかに保存している abc と言う単一のオブジェクトにつけた二つの名前であると言うことです。同じオブジェクトであることを調べるには is を使

ソースコード 6.3　文字列が変更不能であること

```
1 str0 = "abc"
2 str1 = str0
3 print(str0 is str1)
4 str0 = str0 + "def"
5 print(str0)
6 print(str1)
7 print(str0 is str1)
```

います。3 行目で

```
print(str0 is str1)
```

として確かめると、True となります。

4 行目で str0 の後ろに def と文字列を追加しました。str0 という名前を使っていますが、オブジェクトとして別物になっています。つまり、文字列は変更できないのです。このことを、文字列は変更不能 (immutable) といいます。従って 7 行目では、False となります。

課題 6.1　or と not の例題を作成し、動作を確かめなさい。

7　関数: functions

7.1　関数: functions

関数: functions

- 既存のプログラムを再利用する
  - 関数
- ライブラリ

プログラムに必要な機能を、毎回、全てを書くということは考えられません。適切な形で、プログラムを再利用すべきです。その方法の一つが関数 (functions) です。

関数とは、数学関数と同様に、変数 (プログラムでは引数 (arguments) という) を与えると、それを使って何か計算や処理を行って、その結果を返すものです。例えば、三角関数を思い出してください。三角関数は、引数に角度を与えると、それに対応して正弦 sin()

や余弦 cos() の値を返す関数です。

関数は、数学的関数とは限りません。今回の例で出てきた、`type(n)` は、変数 `n` の型を調べる関数でした。あるいは `str(n)` は、引数 `n` を文字列に変換する関数です。

このように、関数は `func(arg)` という形式になっています。引数は複数であることもあります。そのため、「引数並び」ということもあります。

Python には、様々な関数が、言語とともに提供されています。必要な時に、紹介します。

7.2 ライブラリ: Libraries

関数などの有用な機能は、ライブラリというまとまりにして提供されます。初回に、この講義で使用する様々なライブラリをインストールしました。

ライブラリを使用するには、`import` というコマンドを使用します。必要な時に、紹介します。

8 課題

`quiz.ipynb` ある以下の課題を実施しなさい。

二つの変数 $x = 10.0$ と $y = 3.0$ について、x/y、x を y で除した整数部分、x を y で除した余りをそれぞれ計算するプログラムを作成しなさい。また、その結果を書式制御 `f` を用いて文字列化して、印刷しなさい。

9 次回

次回は、条件分岐について学びます。教科書は 5 章です。5 章の内容は、二回に分けて学びます。