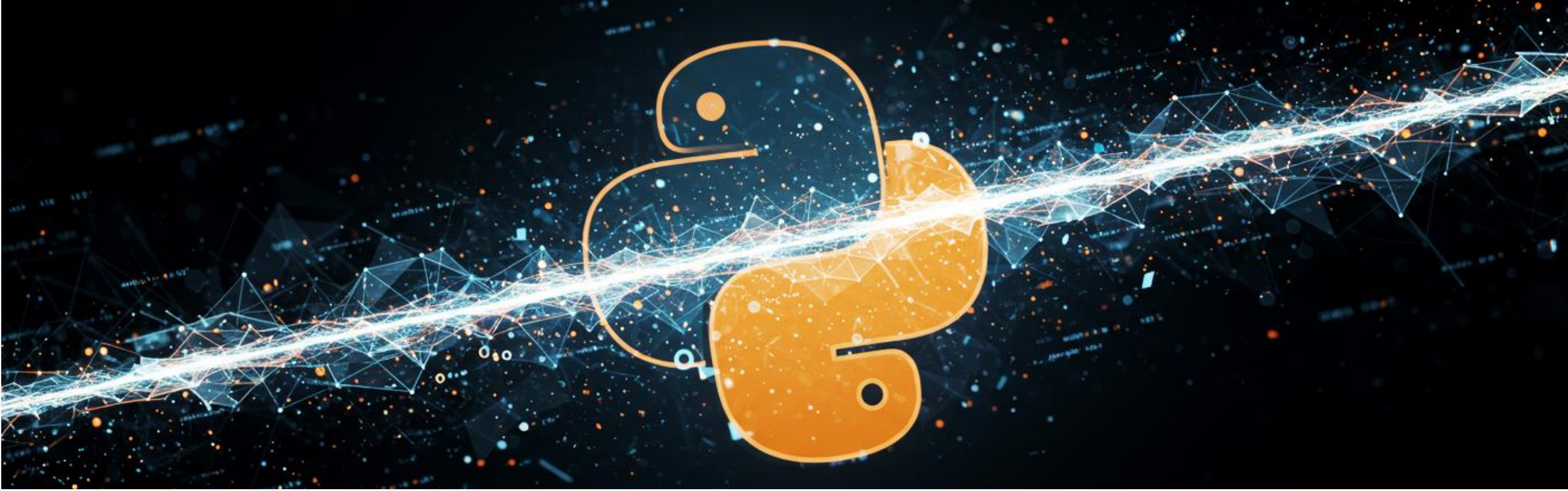

全３回で学ぶPython入門講座
2回目

2025/07/29

Kazuma Sekiguchi

前回のAgenda

- Pythonとは何か
- Pythonの動作環境
- Pythonの記述方法
- 変数、文字列連結
- 条件分岐の書き方
- import文の使い方
- 関数の利用
- ユーザー定義関数の作成

今回のAgenda

- 標準ライブラリーの利用
- ユーザーからの入力方法
- リストの利用
- 繰り返し処理の利用
- 辞書、集合の利用
- ファイルを開く、書き込み保存

Pythonの実行

- VSCodeのターミナルで実行するのが楽
- 他にはターミナルやコマンドプロンプトから実行することも可能
  - 実行するときに実行対象のファイルが存在するフォルダーに移動しておいた方が楽
  - 所定のフォルダーに移動するときは以下のコマンドを使う

```
cd 移動するフォルダー名
```

  - 移動するフォルダー名はパスで指定する
- パスで指定するのが大変であるなら、移動したいフォルダーを参照できるようにしておいて、cdを入力した後で対象のフォルダーをドラッグ＆ドロップすれば入力される

Pythonの記述と実行

- 単一のファイルに記述して保存
- 拡張子はpyとする

```
1回目> python a07.py
```

- ターミナルなどから実行する

python test.py

Pythonで実行する
ファイル名

Pythonで認識されて
いないmacなどの場合

python3 test.py

文字コードによるエラー

- SyntaxError: Non-ASCII characterが表示された場合
  - 文字コードの認識が間違っている可能性
  - Pythonコードの1行目に以下を記述すると直る可能性

```
# -*- coding: utf-8 -*-
```

  - 本来はUTF-8で記述し、実行時もUTF-8のはずなので記述が不要なはずだが、エラーが出た場合は記述してみる
  - 不要でも書いておいても問題は起きないので、書いておくのも手

ユーザーからの入力

- ターミナルなどからユーザーに入力して貰ってPython内で利用することが可能

```
x = input(“入力したい文字を記入してください：")
```

ユーザーが入力した内容が格納される

入力してほしい内容を表示

- 入力された内容は数字であっても文字列として格納される
  - そのまま計算はできない点に注意

ユーザーからの入力で数値

- ユーザーからの入力は全て文字列として受け取る
  - 計算するためには数値として扱って貰う必要がある
  - 型が異なる同士での演算になるため、エラーとなる
- 型変換を行う必要がある
  - 文字列→数値、数値→文字列などのように変数に格納されている値の型を変換する＝型変換（キャスト）
  - 数値→文字列はまず問題が生じないが、文字列→数値は変換できない場合があるので注意
- str(),int(),float()を利用することでキャスト可能

型変換（キャスト）

```
x = “100”
n = 2
print(x + n) #Error
```

```
n = 100
x = “番目”
print(x + n) #Error
```

```
x = “100”
n = 2
print(int(x) + n) #102
```

数値に変換

```
n = 100
x = “番目”
print(str(x) + n) #100番目
```

文字列に変換

```
x = “100”
print(type(x) == str) #True
```

strと比較した結果を表示する

type()で型を取得できる

- そのままだとエラーが生じるが、変換することで目的の動作をさせることが可能
  - 変数の中身がどういう型になっているかを常に把握しておく

リスト

- 1つの変数に複数の値を格納することが可能なもの
- カンマで区切って値を格納することで複数の値を入れることができる
  - 型が異なっているものを入れることも可能（避けた方が無難）
- 末尾のカンマは省略可能
- リストの中身を指定するときはインデックスを利用する
  - インデックス＝リストの順番を示す数字。0から始まる

```
x = [10,20,30,40]
fruits = [‘apple’,’grape’,’orange’,]
print(fruits[0]) #apple
```

リスト内の参照

- インデックスの指定方法はいろいろ
  - :2などのようにコロンを並記するとリストから取り出しをしてリストを作成できる

```
fruits = [‘apple’,’grape’,’orange’,’meron’]
print(fruits[0]) #apple
print(fruits[:2]) #[‘apple’,’grape’]
print(fruits[2:]) #[‘orange’, ‘meron’]
print(fruits[1:3])  #[‘grape’, ‘orange’]
print(fruits[-1]) #meron
```

リストを展開するには？

- for in文を使う
  - リスト内のデータを全て展開してくれる
  - 途中で展開を止めたいならbreak文を、飛ばしたいならcontinueを利用する

```
x = [10,20,30,40]
for n in x:
    print(n)
```

10
20
30
40
が表示される

インデントする

```
x = [10,20,30,40]
for n in x:
    if n == 20:
        continue
    print(n)
```

10
30
40
が表示される

リストの上書き

- リストはインデックスを指定することで上書き（変更）が可能

```
fruits = [‘apple’,’grape’,’orange’,’meron’]
fruits[2] = ‘strawberry’
print(fruits[2]) #strawberry
```

- リスト同士は結合が可能

```
fruits = [‘apple’,’grape’,’orange’,’meron’]
fruits += [‘peach’,’pear’]
print(fruits) #[‘apple’, ‘grape’, ‘orange’, ‘meron’, ‘peach’, ‘pear’]
```

- len（リスト）で要素の数を調べることが可能

タプル

- リストに似ているが、中身の要素を変更できない
  - 要素が1つしか無い場合は末尾にカンマを指定する必要がある
  - タプルを上書きしようとするとエラーが起きる
  - 参照時はリストと同じようして取り出すことが可能

```
fruits = (‘apple’,’grape’,’orange’,’meron’)
print(fruits[1]) #grape
```

関数の戻り値としてのタプル

- 関数の戻り値としてタプルを指定可能

```
def get_date():
    return ‘2023’, ‘6’, ’12’

year, month, day = get_date()
print(year+’/’+month+’/’+day) #2023/6/12
```

タプルとして値を戻す

タプルとして戻ってきた値を変数に順番に格納する

- list(タプル)を使うことでリストに変換可能
  - tuple(リスト）を使うことでリストをタプルに変換可能

辞書（ディクショナリ）

- 変数に複数の値を入れることが可能な点はリストと同じだが、キーと値のペアとして格納することができる

```
user = {‘id’:1,’name’:’Kazuma Sekiguchi’,’age’:42}
print(user[‘id’]) #1
```

- {}を記述してキー:値として指定する
- 参照するときは、キー名を指定することで値を取得できる
  - キーが重ならない場合は追加される
  - キーが重なる場合は、上書きになる

```
user = {‘id’:1,’name’:’Kazuma Sekiguchi’,’age’:42}
user[‘email’] = ‘a@example.com’
```

辞書（ディクショナリ）

- 辞書を展開するときはfor inを組み合わせる

```
user = {‘id’:1,’name’:’Kazuma Sekiguchi’,’age’:42}
for k, v in user.items():
        print(k, v)
for v in user.values():
    print(v)
for k in user.keys():
    print(k)
```

id 1
name Kazuma Sekiguchi
age 42

1
Kazuma Sekiguchi
42

id
name
age

辞書の展開

- 辞書に対して、values()を使うと値だけを取り出すことが可能

```
user = {“1stname”:”富岡”,”2ndname”:”鈴原”,”3rdname”:”綿貫”}
users = list(user.values())#リストに変換して格納する
```

usersには[“富岡”,”鈴原”,”綿貫”]が格納される

- keys()を使えば、キーを取り出すことが可能

map関数

- リスト内の要素に対して同じ関数を適用して戻すことが可能
  - 戻り値がmapオブジェクトという特殊な型になるので、リストに変換する必要はある

```
def calcDouble(x):
        return x * 2
a = [10,20,30,40]

print(list(map(calcDouble,a)))
```

繰り返したい関数を定義

戻り値を指定

list()でリストに変換する

map(繰り返す関数,リスト)として利用

集合（set）

- setは重複のないリストみたいなもの
  - set同士の減算やOR計算、AND計算などが可能
  - 重複を作成しても排除される
  - リストから重複を除外するときなどに便利

```
colorA = set(['red', 'blue', 'green'])
colorB = set(['green', 'yellow', 'white’])
colorC = set (['green', 'yellow', 'white’,’red’,’green’])
print(colorC) #{‘green', 'yellow’, 'white', ‘red’}
print(colorA – colorB) #{'red', 'blue’}
print(colorA | colorB) #{'yellow', 'green', 'white', 'blue', 'red’}
print(colorA & colorB) #{'green'}
```

集合（set）

- 要素を追加するときは、add()を使う

```
colorA.add('white')
```

{'red', 'blue', 'green','white'}

- 要素を削除するときは、remove()を使う

```
colorA.remove('red')
```

{'blue', 'green','white'}

リストにタプル

- リスト内にタプルを入れたり他のものを入れることも可能
- その場合展開がちょっと複雑になる

```
x = [(“果物”,300),(“魚”,500),(“肉”,300)]
for type,price in x:
        print(price)
```

タプルをリスト内に格納

リスト内がタプルなので、2つの変数を指定する

タプルの後側が入ってくるので300, 500, 300の順で表示される

ファイルの扱い

- Pythonからファイルを扱うことが可能
  - 読み込み、書き込み
- ファイルを開いて、処理を行った後でファイルを閉じる
- with文を使うと自動的に処理完了時に閉じることができるので便利
- ファイルを開くときにmodeを指定する必要がある
  - mode=‘r’で読み込みモードになる（デフォルト指定）
  - ファイルが存在しない場合はエラーとなる
- バイナリファイルを開く場合はmodeにbを追記する
  - rbとかwbとかになる

ファイルの扱い

- ファイルを開いてファイル全体を読み込む場合はread()を利用する
  - ファイル全体を文字列として取得可能
- with文を使うことでファイルの処理が簡単になる
  - 開始処理でファイルを開くように指定すれば処理内容を終わらせたら自動的にファイルを閉じてくれる

```
path = ‘sample.txt’
with open(path,mode=‘r’) as f:
        s = f.read()
        print(s)
```

```
with 開始処理 as 変数:
        処理内容
```

ファイルの扱い

- ファイル全体をリストとして取得
  - 最後の要素を除く各要素の末尾に改行が入る

```
path = ‘sample.txt’
with open(path,mode=‘r’) as f:
        s = f.readlines()
        print(s) #[‘1行目¥n’,’2行目’]
```

ファイルの書き込み

- open()の引数modeをwにすることで書き込みモードにすることが可能
  - ファイルが存在しなければ自動的に作成
  - 書き込みができない場合はエラーとなる
  - 存在していれば上書き（追記ではない。追記はa）
  - write()で書き込めるが、文字列のみ書き込める
    - 数値などはstr()で変換する必要がある
    - 改行などもそのまま書き込まれる

```
path_w = ‘sampleWrite.txt'
s = 'New file'

with open(path_w, mode='w') as f:
        f.write(s)
```

ファイルの書き込み

- リストを書き込むときはwritelines()で書き込める
  - 要素がそのまま連結されて書き込まれる

```
colors = [‘Red, ‘Blue', ‘White']

with open(path_w, mode='w') as f:
        f.writelines(colors)
```

- 空のファイルを作成するときはpassを利用する

```
with open(path_w, mode='w') as f:
        pass
```

ファイルへの追記

- ファイルの末尾に追記するときはmodeでaを指定する

```
with open(path_w, mode='a') as f:
        f.write(‘Green')
```

ランダムな数の生成

- 標準ライブラリーであるrandomを使うことでランダムな数を生成することが可能

```
import random


randomNumber = random.randint(1, 100)
#1～100までのランダムな数を返す
```

- 呼び出す度に異なる数を生成して返すことが可能
  - random.seed()で乱数のシード値を指定可能
  - シード値が同一であれば、乱数の結果は同じになる
  - シード値は省略することが可能

チャレンジ

- 以下のプログラムを作成してみてください
  - 答えの例は格納してあります

- ランダムに選ばれた1から100までの整数をユーザーに予想させるゲームを作成してください。ユーザーが正解するまで、プログラムは “高い” または “低い” とヒントを出して一致したら正解と当たるまでに掛かった回数を表示してください。

チャレンジ

- 以下のプログラムを作成してみてください
  - 答えの例は格納してあります

- 収入と支出を登録して、全ての金額の合計と平均収支を表示するプログラムを記述してください。
  収支は支出をマイナス値で記述します。収入：金額、支出：-金額のように記入し、カンマ区切りで1行で入力するものとします。

チャレンジ

- 以下のプログラムを作成してみてください
  - 答えの例は格納してあります

- 曜日別の来店客数を辞書型として適当にプログラムに指定した上で、合計来店客数と平均の来店客数を表示するようにしてください。また、入力された特定の曜日の来店客数を表示するようにしてください。

ありがとうございました。

また次回。