文責：　うね@雪

Python講座

本当に何とでもなる

**目次**

[-はじめに- 3](#_Toc77943866)

[1章.基本編 4](#_Toc77943867)

[1 Pythonってなんだ(飛ばしてもいいよ) 4](#_Toc77943868)

[1.1 Python 4](#_Toc77943869)

[1.2 オブジェクト指向って何！！ 5](#_Toc77943870)

[2 開発環境(なんなら一番面倒くさい) 6](#_Toc77943871)

[2.1 開発環境とは 6](#_Toc77943872)

[2.2 なにがあって,なにがいいの？ 6](#_Toc77943873)

[2.3 インストール方法 & 使い方(Anaconda) 6](#_Toc77943874)

[2.4 インストール方法 & 使い方(Python + VScode) 6](#_Toc77943875)

[3 モジュール？ライブラリ？ 7](#_Toc77943876)

[3.1 言葉の説明 7](#_Toc77943877)

[3.2 インストール方法 7](#_Toc77943878)

[3.3 使い方(import? from?) 7](#_Toc77943879)

[4 型の話 8](#_Toc77943880)

[4.1 色んな型 8](#_Toc77943881)

[4.2 Pythonにおける型の扱い(型変換) 8](#_Toc77943882)

[5 ごめん、結構ごっそりまとめる 9](#_Toc77943883)

[5.1 変数について 9](#_Toc77943884)

[5.2 Pythonのインデント(Tabキー) 9](#_Toc77943885)

[5.3 リスト！ 9](#_Toc77943886)

[5.4 辞書も便利 9](#_Toc77943887)

[5.5 関数(ちょっとだけ) 9](#_Toc77943888)

[5.6 条件(真 と 偽) 9](#_Toc77943889)

[5.7 分岐処理(if / else if / else) 9](#_Toc77943890)

[5.8 ループ処理(for / while) 10](#_Toc77943891)

[5.9 <発展>ループ処理(enumerate) 11](#_Toc77943892)

[6 関数作ろー！ 12](#_Toc77943893)

[6.1 関数の構造 12](#_Toc77943894)

[6.2 実際に作ってみよ 12](#_Toc77943895)

[6.3 軽－く練習問題 12](#_Toc77943896)

[7 クラスって、、？メソッドと関数、、？？ 13](#_Toc77943897)

[7.1 インスタンスという概念(わたあめ) 13](#_Toc77943898)

[7.2 ひとまずクラスを作ってインスタンス化 13](#_Toc77943899)

[7.3 \_\_init\_\_なにそれおいしいの？ 13](#_Toc77943900)

[7.4 クラス変数、インスタンス変数 13](#_Toc77943901)

[7.5 メソッドと関数って何が違う 13](#_Toc77943902)

[7.6 <発展>継承 13](#_Toc77943903)

[7.7 <発展>抽象クラス 13](#_Toc77943904)

[8 例外処理って便利なんだよ 14](#_Toc77943905)

[8.1 tryの使い方 14](#_Toc77943906)

[8.2 実際に使ってみる(AttributeError回避) 14](#_Toc77943907)

[2章.応用編 15](#_Toc77943908)

[おまけ 16](#_Toc77943909)

[ エラーコード早引き 16](#_Toc77943910)

[ よく使うメソッド 16](#_Toc77943911)

[ リストに対するメソッド 16](#_Toc77943912)

[ 文字列に対するメソッド 16](#_Toc77943913)

[ よく使うライブラリ 16](#_Toc77943914)

[ numpy 16](#_Toc77943915)

[ matplotlib 16](#_Toc77943916)

[ pandas 16](#_Toc77943917)

[ バージョン管理(git) 16](#_Toc77943918)

[1. バージョン管理の考え方 16](#_Toc77943919)

[2. gitとは 16](#_Toc77943920)

[3. ブランチ 16](#_Toc77943921)

[4. ちょっと全部は無理かもしれない 16](#_Toc77943922)

-はじめに-

この文書が僕の知りえないところまで広まってないことを少しだけ祈っておきます。(ハズいから)

どうも、改めまして一応本名は伏せておきます、うね@雪です。はじめにっても書くことが特にありません。「この物語はフィクションです。実在の人物や団体などとは関係ありません。」って言っといたら間違えても怒られないかな。めっちゃ実在する言語だけど。

さて、なぜこんなものを作るに至ったか。まぁ暇だからが8割なんですけど、残りの2割は色々です。現時点でPython全く知らんって人は今後Pythonに触れる機会はないと思いますが、意外と知っておいて損はない気がします。便利なんだよ、プログラミングって。一方、Pythonやり始めたけど、いまいち何するか分からんって人や、結構もう知ってるよって人。そんな人たちにもなんかの手助けになれないかなーって思って書いてます。なんと初心者から中上級者まで、全年齢対象でございます。Gって付けても怒られない気がします。とはいえ一応、大学生推奨って感じです。でもR18って付けたら怒られそうです。

この時代、小学生でもプログラミングに触れるようになっています。さすがに小学生ががっつりjava, Pythonを扱うのは珍しいですが、概念だけでも幼いころにやっていれば、中学生高校生ではもう十二分にコードが書けるようになります。ぶっちゃけ恐ろしい時代です。そんな化け物どもに対抗するためにはこちらも武装しておかねばなりません。今現在、ほとんどの企業で、ほとんどの部署で、ほとんどの業界で、コンピュータは使われています。ってことは、どこででもプログラミングってできるわけですね。どれだけプログラミングから離れてても導入される危険性があるわけです。そんな万が一に備えるためにも、一応触れてみてもいいんじゃないだろうかと思うわけです。

そうなって勉強しようってなった時、世の中の参考書を否定するわけじゃないですけど、どれも難しいんすよ。いやマジで。怖いもん。なので、この資料は、できーるだけ簡単に、できーるだけ気を抜いて読んでもらえるよう書いてるつもりです。分かんない箇所があれば、個別対応もできるので、ぜひぜひ一読していただけると嬉しいです。そんな感じでよろしくお願いします。

1章.基本編

1. Pythonってなんだ(飛ばしてもいいよ)
   1. Python

Pythonはオブジェクト指向型プログラミング言語の代表的な言語です！……と言われて”はい、そうね”って流してもらうのが一番平和だと思います。オブジェクト指向、これは後々さっくり説明しますが、別に分からなくても特に問題ないです。その割に色んな入門書では、さも重要そうに書くんです。いや入門で書くなよって思う。

他にPythonってなにさって言われるとなんだろうな。僕の勝手な感覚ではjavaとCを足して2で割った上に角を丸くした、、って感じですね。ベースはjavaみたい、でも計算はC言語を用いてるっぽいです。なんかそんなもんです。

じゃあ、そもそもプログラミング言語ってなんなんだ。元々コンピュータってのは0, 1しか受け付けない人なんです。普通に考えてスワヒリ語圏の人に日本語で喋っても伝わらないよね。僕らの言葉とコンピュータとの仲介(翻訳)をしてくれるのがプログラミング言語です。ほなjavaだとかCだとかPythonだとかってなんやねんと。ざっくり言っちゃえば英語と日本語とスペイン語みたいな感じ。言ってることは一緒だけど文法が違うの。だから、専門的な拘りが無ければ基本的にどれを使っても大丈夫なんです。一応、それぞれ得意不得意があったり翻訳方法が違ったりします。

さて、有名な話ですが、AIといえばPythonという風潮があります。なぜかと言いますと、Pythonでは豊富な機械学習ライブラリ(後述)が公開されているからです。だから僕はPython。ってか、単純に文法が簡単なんだよね。javaを勉強したことがある人は分かると思うけど、異常にオブジェクト指向なの。だから結構しっかり理解して覚えて書かないといけないし、読むのも難しい。けど、Pythonって割とてきとーに書いても動いてくれるんです。……ほんとだよ？

そんな愛すべきPythonのお話、始まり始まりー。

* 1. オブジェクト指向って何！！

実は、僕もオブジェクト指向っていうの最近までよく分かんなかったんですよねー。実務でようやくなんか理解した感じ。

よく参考書にある記述がこれです。「オブジェクト指向とは、すべてをオブジェクトとして扱うこと」……は？(ガチギレ)　説明ってかもはや言葉そのままやないか！　発酵を「発酵させること」って書いてるのと一緒。発酵ってなんやねん。

そんな感じでずーっと分かんなかったんですよね。さて、では僕なりの説明をしていきます。厳密には間違ってるかも、ごめんね。

まず、オブジェクト。これはなんだ。言葉の意味そのままだと“物”ですね。その訳はいったん置いといてください。ね。今からオブジェクトとは、“操作できるもの”だと思ってください。例えば、スマホ。僕らはスマホで、文字を打ったり、動画を検索したり、ゲームをしたりできます。僕たちは、スマホを操作できるわけです。現実世界のスマホはオブジェクトにあたるんですね。もう少し考えてみましょう。じゃあねー、テレビ！　テレビも、電源を入れる、チャンネルを変える、音量を上げる、などなど操作ができます。つまり、テレビもオブジェクトです。

オブジェクトの概念が何となくでも分かりました？　現実世界ではスマホやテレビなど、操作できるものがオブジェクトです。(気体も一応操作できるのでオブジェクトなのかな)　じゃあ、オブジェクト指向ってなんだと。ここで最初の言葉“(プログラミングにおける要素)すべてをオブジェクトとして扱う”の意味がちょっと分かったんじゃないでしょうか。

プログラミングにおける要素ってのは、要は変数のことですね。変数ってのは、まぁ“なんか入ってるもの”ぐらいでいいです。Pythonを例にして、a =‘test’と変数を設定します。今、aには文字列(test)が入っています。オブジェクト指向では、aを操作できるものとして扱います。例えば、この文字を大文字にする(TEST)、一番左の文字を消す(est)、特定の文字を消す(es(tを削除))、などの操作ができるよう言語を作ってるよってのがオブジェクト指向です。

オブジェクト指向、要は言語の設計図ってことですね。

1. 開発環境(なんなら一番面倒くさい)
   1. 開発環境とは

これを作るにあたって僕が一番面倒くさいと感じるのがこの章です。めんどうすぎて、一番最後に書いてます。

1-1で少し触れましたが、プログラミング言語ってのは翻訳機です。となると、翻訳機に僕らの言語を渡してあげなくてはいけません。その橋渡しをしてくれるのが開発環境ってやつです。要はコードを書くところです。ただ、何が面倒くさいって、なんか一番不具合が多いんですよねー……。しかもこの機種はエラー出ないのにこっちの機種ではエラーが出るとか、環境依存してたりもします。えぐい。

一応、僕の環境におけるインストール画面を踏まえつつ説明していきます。基本的にこの通りにやれば大丈夫だと思います、、。

* 1. なにがあって,なにがいいの？
  2. インストール方法 & 使い方(Anaconda)
  3. インストール方法 & 使い方(Python + VScode)

1. モジュール？ライブラリ？
   1. 言葉の説明
   2. インストール方法
   3. 使い方(import? from?)
2. 型の話
   1. 色んな型
   2. Pythonにおける型の扱い(型変換)
3. ごめん、結構ごっそりまとめる
   1. 変数について
   2. Pythonのインデント(Tabキー)
   3. リスト！
   4. 辞書も便利
   5. 関数(ちょっとだけ)
   6. 条件(真 と 偽)
   7. 分岐処理(if / else if / else)

いわゆるif文ってやつですね。

|  |
| --- |
| if 条件1:  条件1が真なら実行  else if 条件2:  条件2が真なら実行  else:  どの条件にも当てはまらなかったら実行 |

こんな感じ。上の例では条件が2つだけでしたが、else ifを続けていくことで、いくらでも条件は増やせます。もちろん、else ifやelseは省いても大丈夫です。僕がよく使う条件式は「リストの要素が無い」ですが、これはこんな感じで書けます。

|  |
| --- |
| a = []  if not a:  a.append(“test”)  print(a)  else:  print(a) |
| ['test'] |

リストのところでは説明しませんでしたが、リストは“要素があると真”、“要素が無いと偽”を出してくれるんですね。よって、「要素が無い」というのを条件にするには、notを付けてやればうまく判定できます。リストが空だと困ることが多いので、結構よく使います。

* 1. ループ処理(for / while)

ループ処理ですが、書き方自体は簡単です。

|  |
| --- |
| for 変数 in イテレータ:  ループさせたい処理 |

とまぁ、書いてみたもののなかなかややこしいところがありますね、、。イテレータってのは“要素を持つもの”って考えるのが一番簡単です。リストもイテレータの一種なので、イテレータ＝リストって捉えてもいいと思います。

for文はイテレータの最初の要素から順に、変数に代入してくれます。for文の中でその変数を用いて処理をすることが多いですね。

実例を用いつつ説明していきます。

<リストをループさせるとき>

|  |
| --- |
| my\_list = [“ワン”, “ツー”, “さん”, “し”]  for i in my\_list:  print(i) |
| ワン  ツー  さん  し |

リストは簡単ですね。要素を最初からそのままiに代入してくれます。

<指定回数ループさせるとき>

|  |
| --- |
| for i in range(4):  print(i) |
| 0  1  2  3 |

上の例は4回ループさせるときの使い方です。rangeって関数に回数を入れてあげるだけです。「for i in 4:」ってダメなの？ってなりますが、ダメなんです。4はint型なのでイテレータじゃないんです。range関数は4(int)に対して[0, 1, 2, 3]っていうイテレータを作ってくれるんです。なので、出力が“0, 1, 2, 3”になってますね。

<辞書をループさせるとき>

|  |
| --- |
| my\_dict = {1: “ワン”, 2: “ツー”, 3: “さん”, 4: “し”}  for i in my\_dict:  print(i) |
| 1  2  3  4 |

これが辞書を単純にループさせたときの挙動になります。……うん、なにこれ。そうなんです。辞書をそのままforに突っ込むとキーだけが出てくるんです。まぁ、変数一個だし、キーか要素だよねって話。じゃあ、次！！

<辞書をループさせるとき②>

|  |
| --- |
| my\_dict = {1: “ワン”, 2: “ツー”, 3: “さん”, 4: “し”}  for i in my\_dict.keys():  print(i) |
| 1  2  3  4 |

無意味！！！

はい、.keys()メソッドでキーが取り出せるんですけど……for文で使うなら要らないっすね。

<辞書をループさせるとき③>

|  |
| --- |
| my\_dict = {1: “ワン”, 2: “ツー”, 3: “さん”, 4: “し”}  for i in my\_dict.values():  print(i) |
| ワン  ツー  さん  し |

これはまぁそこそこ使うかな、.values()メソッドで辞書の要素を取り出すことができます。よく“s”を忘れて怒られます、、。

<辞書をループさせるとき④>

|  |
| --- |
| my\_dict = {1: “ワン”, 2: “ツー”, 3: “さん”, 4: “し”}  for key, value in my\_dict.items():  print(key, value) |
| 1 ワン  2 ツー  3 さん  4 し |

最後に、これがいっっっちばん使う。.items()メソッドを使うと、2変数に分解して、キーと要素を取り出してくれます。

* 1. <発展>ループ処理(enumerate)

1. 関数作ろー！
   1. 関数の構造
   2. 実際に作ってみよ
   3. 軽－く練習問題
2. クラスって、、？メソッドと関数、、？？
   1. インスタンスという概念(わたあめ)
   2. ひとまずクラスを作ってインスタンス化
   3. \_\_init\_\_なにそれおいしいの？
   4. クラス変数、インスタンス変数
   5. メソッドと関数って何が違う
   6. <発展>継承
   7. <発展>抽象クラス
3. 例外処理って便利なんだよ
   1. tryの使い方
   2. 実際に使ってみる(AttributeError回避)

2章.応用編

おまけ

* エラーコード早引き
* よく使うメソッド
* リストに対するメソッド
* 文字列に対するメソッド
* よく使うライブラリ
* numpy
* matplotlib
* pandas
* バージョン管理(git)
  1. バージョン管理の考え方
  2. gitとは
  3. ブランチ
  4. ちょっと全部は無理かもしれない