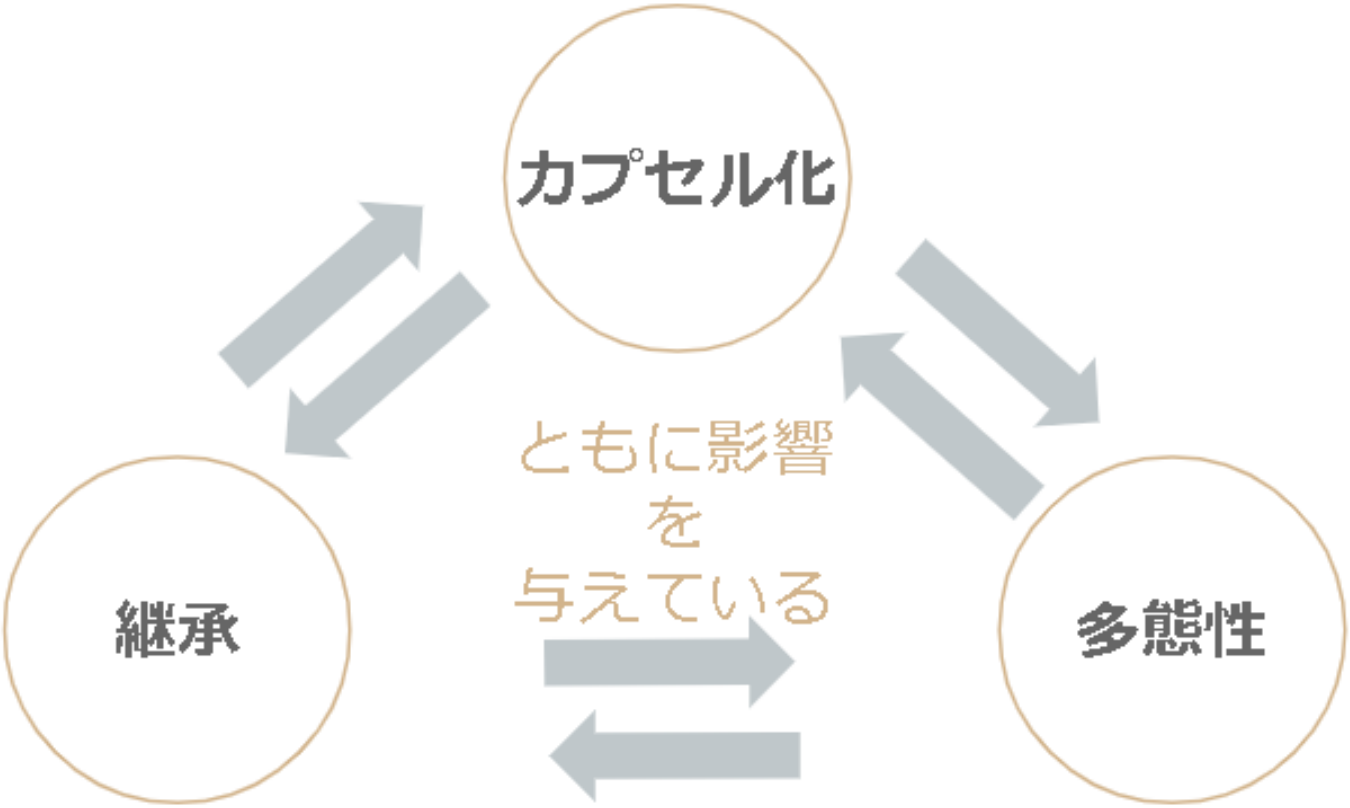
Pythonとは?

Pythonは、1991年にオランダ人のグイド・ヴァンロッサムさんが開発したプログラミング言語です。

Pythonという名前は、ヴァンロッサムさんが好きだった『空飛ぶモンティ・パイソン』からとりました。

そのため、Pythonという名前は、英単語としての意味「ニシキヘビ」とは関係ないですが、アイコンにヘビが使われることが多いです。



Pythonはオブジェクト指向言語です。

オブジェクト指向言語とは、オブジェクト指向開発という「データと処理」を1セットとしてプログラムを組み立てていく開発手法に適した言語のことです。

Pythonができるようになると、人工知能開発、データ分析、Webアプリケーション開発ができるようになります。

学びやすい言語

Pythonは学びやすい言語です。

Pythonは「文章を読むようにわかりやすいコード」を目標に作られたプログラミング言語です。

実際に、読みやすく、書きやすい言語となっています。

例を1つあげてみましょう。

「Hello,world」という文字を表示させるコードを、C言語とPythonの両方で書いて比較します。

"#include <stdio.h>

int main() {

    printf("""Hello World\n""");

}"

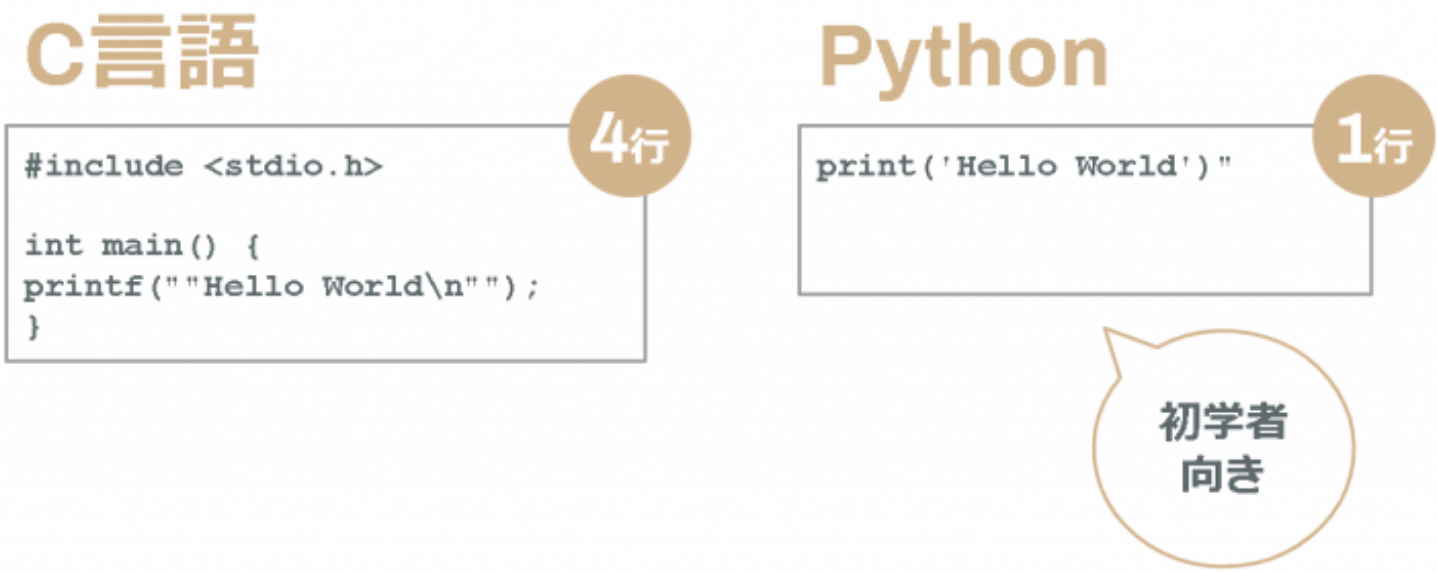
C言語では「Hello World」と表示させるのに、4行が必要になります。

次に、Pythonで書いてみましょう。

print('Hello World')"

たったこれだけです。

Pythonだと1行で表示させることができるのです。



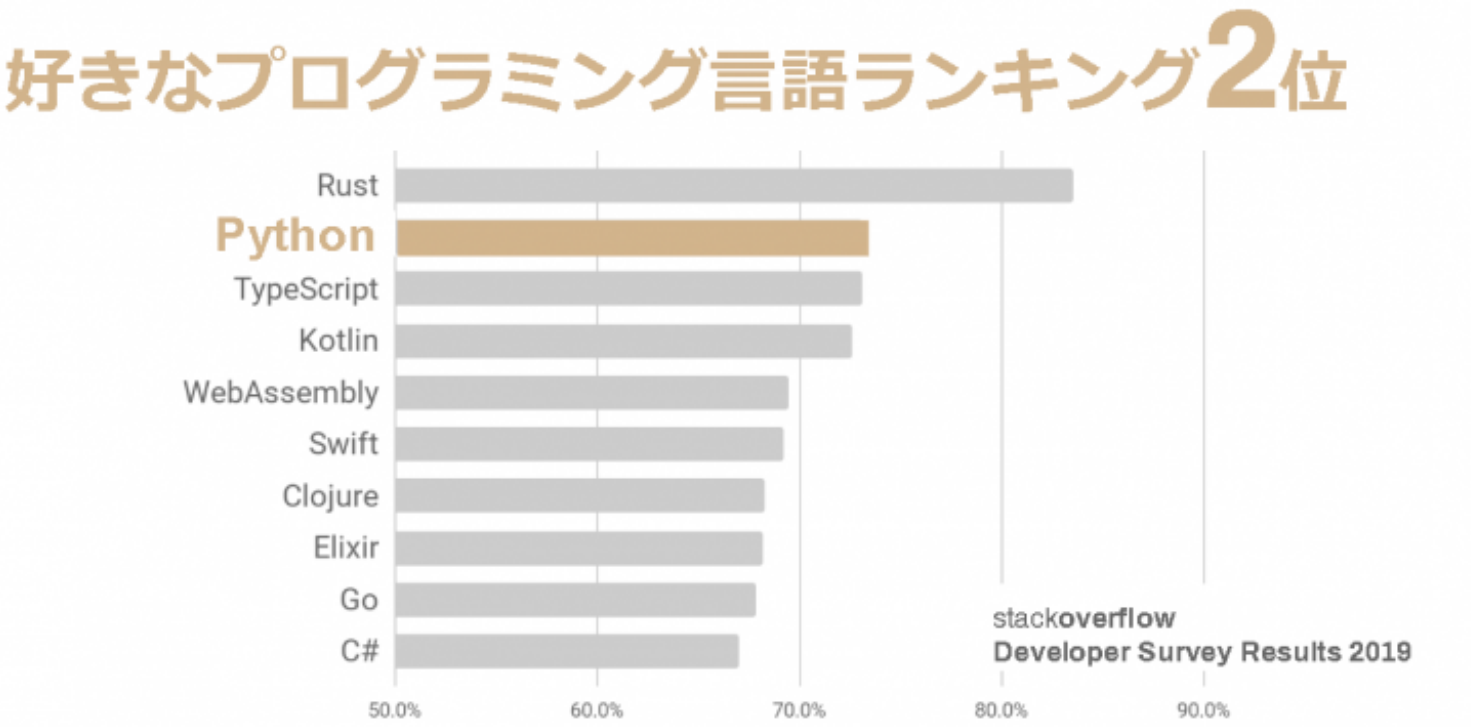
Pythonは、読みやすく、書きやすいため、プログラミング初学者に向いているといえます。

ちなみに、Pythonのような読みやすく書きやすい言語のことをスクリプト言語といいます。

人気言語

Pythonは人気言語です。

2019年のstackoverflowの調査で、好きなプログラミング言語ランキング2位に選ばれています。



また、世界規模のテクノロジー起業のGoogleでは、社内の標準プログラミング言語として、Pythonを採用しています。

Youtube、Facebook、Instagram、Netflixなどのインターネットサービスでも Pythonが活用されています。

人工知能を作れる

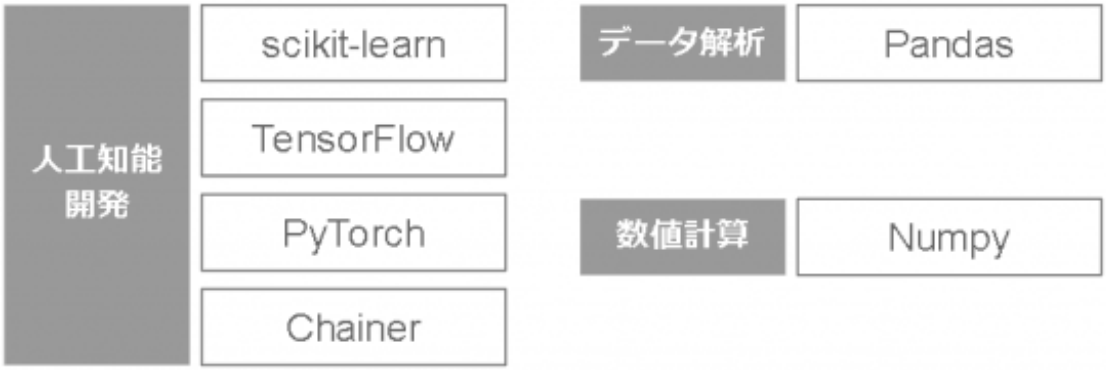
Pythonを使えるようになると、人工知能を作ることができます。

Pythonには、人工知能開発のための「scikit-learn」「TensorFlow」「PyTorch」「Chainer」、データ解析を支援する「Pandas」、数値計算の「NumPy」など、人工知能の開発に効率的にするライブラリが豊富に存在します。そのため、Pythonは人工知能開発には欠かせない言語となり、昨今のPython人気に火をつけました。

Python

人工知能を作れる

ライブラリが豊富



環境構築に必要なもの

環境構築とは、プログラムを書いたり、実行できたりする環境を自分のコンピュータに整えることをいいます。

環境構築のために、VisualStudioCode本体、VisualStudioCodeのPython拡張機能、anacondaの3つをインストールします。

インストールするものを1つずつ簡単に説明します。

まず、Visual Studio Codeについてです。

Visual Studio Codeとは、マイクロソフトが開発したプログラムを書くためのエディタです。

VScodeとも呼ばれています。

VScodeは、Windows、macOS、Linux、色々なOSで使うことができます。

また、Java、C言語、pythonなど有名どころのプログラミング言語を書くことができます。

VScodeは、好きな開発環境の1位に選ばれている人気の開

発環境です。

キノコードのレッスンでは、設定などで迷うことなく言語の学習に集中できるように、基本的にはVScodeを開発環境として使っていきます。

インストールするものは、概要欄にURLを貼っておきます。

次に、anacondaです。

anacondaは、Pythonのディストリビューションです。

anacondaには、Pythonで使われるさまざまなライブラリが入っている便利なものです。

anacondaのインストール方法

それでは、anacondaのインストールをしていきます。

Googleで「anaconda download」と検索すれば出てきます。

これが公式サイトです。URLは概要欄に記載しています。

downloadをクリック

「Python3系」のanacondaがありますのでダウンロードしましょう。

（ダウンロードには数分かかります）

ダウンロードが完了したら、ダウンロードフォルダを開きましょう。

クリックをしてインストール。

次々、進んでいきましょう。

インストールが完了しました。

「閉じる」をクリックします。

インストールに使ったファイツは削除していよいです

3つの基本構造とは?

プログラムではインターネットのサービスや人工知能、スマホアプリ、ゲームなどいろいろなものを作ることができます。

そう聞くと、プログラムでは複雑なことをしているイメージをもつかもしれません。

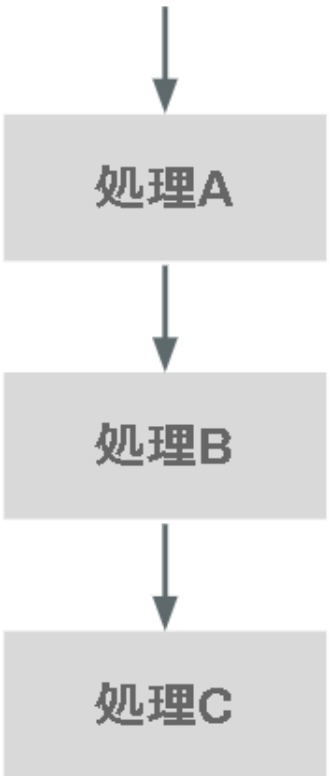
でも、プログラムの動きはシンプルです。

動きは3つだけです。

「順次進行」「条件分岐」「繰り返し」です。この3つの動きのことをまとめて、プログラムの基本構造と言ったり、制御構造、制御フローと言ったりします。

この3つの基本構造を使えば、複雑なプログラムを作ることができます。
そして、この3つは、どのプログラミング言語にもあります。

順次進行



まずは、「順次進行」です。

順次進行とは、プログラムが書かれている上から順に処理をしていくというプログラムの構造です。

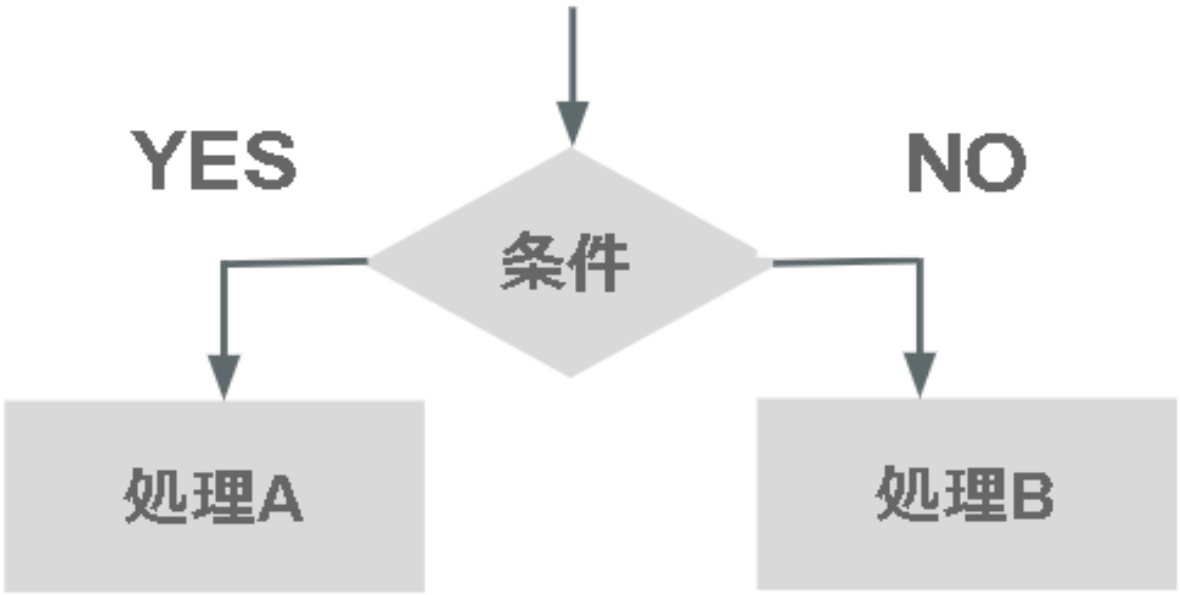
プログラムの最も基本的な動きになります。

プログラムのソースコードの記述が、上から順に、処理A、処理B、処理Cと記述されていたら、処理も上から順に、処理A、処理B、処理Cと処理されていきます。

例えば、ソースコードが「おはよう」「こんにちは」「こんばんは」とパソコンの画面上に表示させるプログラムだったとします。

プログラムを実行すると、上から順に、「おはよう」「こんにちは」「こんばんは」と表示されます。

条件分岐

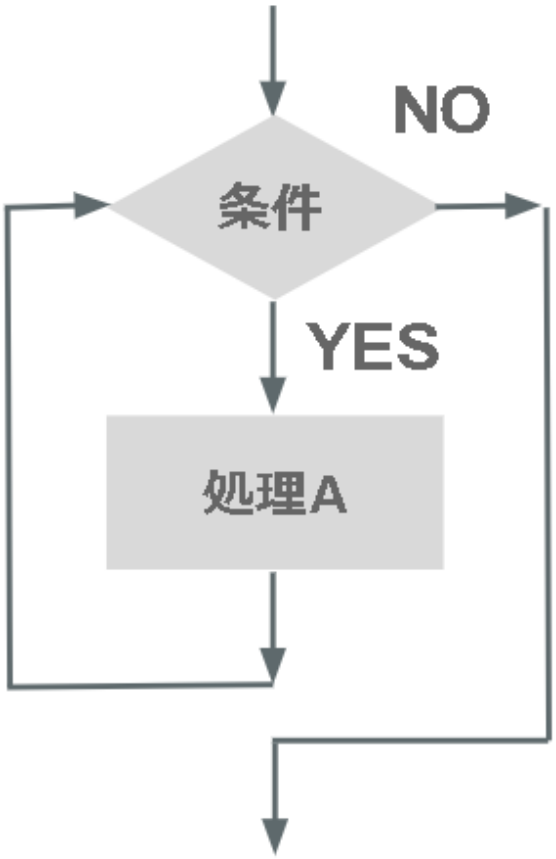


次に、「条件分岐」です。

条件分岐とは、特定の条件のときはAという処理、そうじゃないときはBという処理をするプログラム構造です。

例えば、あるデータの値が20以上なら「おとな」と画面上に表示させて、20未満なら「こども」と表示させるといった処理ができます。

繰り返し



繰り返しとは、決まった回数や条件を満たすまで同じ処理を繰り返すプログラム構造です。

繰り返しは、反復処理といったりもします。

例えば、「こども」という文字を繰り返し20回表示させたい場合などに使います。

他にも、あるデータに1ずつ足していき、そのデータが20未満であれば「こども」を表示させる。20以上になったら、繰り返し処理を終わらせるということでできます。

Pythonファイルの作成

このレッスンでは、パソコン上に挨拶の「Good morning（おはよう）」「Good afternoon（こんにちは）」「Good evening（こんばんは）」と表示させるプログラムを記述・実行してみましょう

まず、VScodeを起動します。

これがファイル管理のための「エクスプローラ」です。クリックします。

「フォルダを開く」をクリック

デスクトップに「MyPython」というファルダを作りましょう。

選択して開く。

MyPythonのフォルダが選択されています。Pythonファイルを作ってみましょう。

英語で挨拶は、Greetingといいます。ファイル名を「Greeting.py」とします。

これでPythonファイルの作成はおわりです。

このレッスンでは、Pythonのプログラムの書き方や、実行のやり方を把握してもらうだけでよいです。

プログラムの意味については、あとに続くレッスンで説明するので、今はプログラムの意味は、「ルール」や「きまり」だと思って進めてください。

こうやって書くんだーって程度でよいです。

早速、コンピューターに挨拶を表示させるコードを書いていきましょう。

```
print("Good morning")

print("Good afternoon")

print("Good evening")
```

実行

プリント、カッコ、ダブルクオーテーション、グッドモーニングです。これでおわりです。

printがコンピュータに文字列や数値を表示させる関数です。丸括弧の中に表示させたい文字列などを記述します。

数値を表示させるときは、ダブルクオーテーションはいりませんが、文字列を表示させるときはダブルクオーテーションがいります。

続けて、「Good afternoon」「Good evening」も順に書いていきましょう。

これでファイルを保存します。

保存方法は、ファイル→保存です。

保存ができると、この白丸が消えて×になります。

保存してみましょう。×になりました。

ちなみに、保存は、ショートカットでコマンドとSでも保存できます。

ファイルを実行して、文字列を表示させてみましょう。

ターミナルから実行します。

ターミナルを表示させるには、表示→ターミナルです。

実行方法は、pythonと書いて、pythonのファイル名です。
まずpythonと書きます。ファイル名は、Greeting.pyなので、これを記述します。
エンターで実行です。
すると、ターミナルに「Good morning」「Good afternoon」「Good evening」という順番に表示されました。
では、順次進行されるのか順番に入れ替えてみましょう。

```
print(""Good evening"");

print(""Good afternoon"");

print(""Good morning"");
```

保存しましょう。コマンドとSで保存してみます。保存できました。
実行してみましょう。
上からの順番に「Good evening」「Good afternoon」

「Good morning」となっています。

先ほどとは違う実行方法でやります。

プログラムを記述する箇所で右クリック。

「ターミナルでPythonファイルを実行」をクリック

上からの順番通り「Good evening」「Good afternoon」

「Good morning」と表示されました。

順次進行されています。

変数とは？

まず、変数について説明します。

変数は、例えるなら、「箱」のことです

変数があることで、文字や数字などのデータを変数に入れておくことができます

また、変数を必要な時に取り出すことができます。

変数にデータを入れることを「代入」といい、取り出すことを「参照」といいます。

変数には名前をつけることができ、変数名といいます。

変数を作ることを「変数を宣言する」といいます。

初めて変数にデータを入れることを「変数の初期化」といいます。

Pythonの変数の宣言方法

Pythonのコードを書いていきましょう。

```
num = 1
print(num)
```

このコードは、変数numに整数の1を代入して、その変数の中身を表示させるプログラムです。

ここで、変数numに1という整数を代入しています。

変数へ代入するときは、イコールを使います。

これで変数に1を代入できます。

Pythonの文字列や変数の表示方法

変数に1が代入されているか、変数を表示させてみましょう。

前のレッスンでも使いましたが、文字列や変数の中を参照するには、print関数を使います。

前のレッスンではダブルクォテーションで文字列を囲いましたが、今回は変数の中を表示したいので、変数をそのま

ま記述します。

実行してみましょう。

実行結果：

```
1
```

「1」という整数が表示されました

numに「1」が代入されていることがわかります。

Pythonの変数名のルール

変数名に使える文字には、ルールがあります。

変数名には、アルファベット、数字、アンダースコア(_)が使えます。

変数名がつけられるか、あるいは、エラーになるか試してみましょう。

```
num = 1
num01 = 2
num_01 = 3

print(num)
print(num01)
print(num_01)
```

numのあとに01の数字をつける変数を作ってみます。

この変数には2を代入します。

numのあとにアンダーバー、そのあとに01の数字をつけてみましょう。

この変数には3を代入します。

それぞれの変数を表示させてみましょう。

実行結果：

```
1
2
3
```

エラーにならずに、「1、2、3」が表示されました。

変数名は、数字から始めることができません。

また、アンダーバー以外に記号を使うことができません。

コードを書いて試してみましょう。

```
num = 1
num01 = 2
num_01 = 3
num$01 = 4
num-01 = 5
01num = 6
```

```
print(num)
print(num01)
print(num_01)
print(num$01)
print(num-01)
print(01num)
```

numのあとにドルマークをつけてみましょう。

また、numのあとにハイフンをつけてみましょう。

最後に、変数名の前に数字をつけてみましょう。

変数名が赤い文字になっています。エラーです。

念の為、実行をしてみます。

実行結果：

```
    File "", line 4
      num$01 = 4
          ^
SyntaxError: invalid syntax
```

エラーになりました。

```
num = 1
num01 = 2
num_01 = 3
# num$01 = 4
# num-01 = 5
# 01num = 6

print(num)
print(num01)
print(num_01)
# print(num$01)
# print(num-01)
# print(01num)
```

こちらについてはエラーになるので、コメントアウトしておきましょう。

コメントアウトとは、記述したプログラムを処理させない

ようにすることです。

コメントアウトはシャープを記述することでできます。

コメントアウトは、ショートカットでもできます。

macの場合だと「command + /（スラッシュ）」、

Windowsの場合だと「［Ctrl］＋［/］」でできます。

この状態で実行してみます。

実行結果：

```
1
2
3
```

エラーの部分はコメントアウトしたため、エラーにならず実行されました。

大文字と小文字は区別されます。

```
NUM = 1
Num = 2

print(NUM)
print(Num)
```

すべて大文字のNUM、最初だけ大文字のNumを作って変数を代入させてみましょう。

それぞれ1と2を代入してみます。

もし仮に、大文字と小文字が区別されないのであれば、両方ともに2と代入されるはずです。

実行してみましょう。

実行結果：

```
1
2
```

別々の数字が表示されました。

予約語は変数名にすることができません。

予約語とは、「return」「class」「for 」「while」など、プログラミング言語ですでに役割が決まっている単語のことをいいます。

「return」という変数をつくってみましょう。

```
return = 10

print(return)
```

実行結果：

```
        File "", line 1
          return = 10
```

```
    ^
SyntaxError: invalid syntax
```

赤い波線が表示されています。エラーです。

データ型とは？

データ型とは、変数に入れるデータの種類のことです。

この動画では、数値型、文字列型、ブール型について説明します。

ちなみに、Pythonでは、変数にデータを入れるときに、データ型を指定する必要がありません。

Pythonが自動的にデータ型を判断してくれるからです。

このようなプログラミング言語のことを動的型付け言語といいます。

動的型付け言語には、Python、Ruby、Python、PHPなどがあります。

対して、変数にデータを入れる時にデータ型を指定する言語のことを、静的型付け言語といいます。

静的型付け言語には、C言語、Java、Kotlin、Goなどがあります。

整数 型

それではまず、最初に、数値型について説明します。

数値型には、整数の「int型」、小数点の「float型」があります。

Pythonのプログラムを書いて、みていきましょう。

```
num01 = 123
num02 = 1.23

print(num01)
print(num02)
```

num01には、整数の123を代入。

num02には、小数点の1.23を代入します。

実行してみましょう。

実行結果：

```
123
1.23
```

表示されました。

Pythonのデータ型は、type（タイプ）を使用して確認する事が出来ます。

typeの丸括弧の中にnum01を入れて、それをprintでくくりましょう。

```
num01 = 123
num02 = 1.23

print(type(num01))
print(type(num02))
```

実行してみます。

実行結果：

```
int
float
```

intとfloatが表示されました。

文字列型

次に、文字列型について説明します。

文字列型は「string型」ともいいます。

string_aという変数を作って、そこへ文字列を代入してみましょう。

```
string_a = "Hello,World!"
```

```
print(string_a)
print(type(string_a))
```

文字列はシングルクォーテーション、またはダブルクォーテーションでくくります。

今回はダブルクォーテーションで「Hello,World!」の文字列をくくります。

string_aの中身を表示させ、あわせて、データの種類を表示させてみましょう。

「Hello,World!」と、データ型のStringが表示されるはずです。

実行してみましょう。

実行結果：

```
Hello,World!
str
```

予想通り「Hello,World!」と 「String」が表示されました。

ブール型

最後に、ブール型です。

ブール型は、Boolean型（ブーリアン型）ともいいます。

ブール型は、TrueまたFalseの2つのうち、どちらか1つを持つ型です。

TrueとFalseとはなんでしょうか？

例えば、10と1では10の方が大きいです。

つまり、「10>1」は正しいです。

これを変数に代入するとTrueが入ります。

逆に「10<1」は誤りです。これを変数に代入すると、Falseが入ります。

コードを書いてみていきましょう。

```
a = 10
b = 1
```

```
bool01 = (a > b)
print(bool01)
print(type(bool01))
```

aという変数に10を代入。bという変数に1を代入しましょう。

a>bは正しいです。

結果をbool01に代入してみましょう。

Trueが代入されいるはずです。

bool01のデータ型も確認してみましょう。

実行してみます。

実行結果：

```
True
bool
```

「True」と、ブール型である「boolean」が表示されました。

不等号を逆にすると誤りなのでFalseが代入されているはずです。

```
a = 10
b = 1

bool01 = (a < b)
print(bool01)
print(type(bool01))
```

実行してみましょう。

実行結果：

```
False
bool
```

Falseとブール型の「boolean」が表示されました。

配列とは？

リストとは、複数のデータを格納することができるデータ型です。

変数のレッスンで、変数は「箱」と説明しました。

例えるなら、変数は1つのデータしかいられない「箱」なのに対して、リストは複数のデータを入れることができる「ロッカー」です。

リストの1つ1つの箱のことを要素といいます。

リストのそれぞれの要素には、場所の情報が割り当てられています。

データが住んでいる住所のようなものです。

この住所には、インデックスという番号が割り当てられています。

一番左のインデックスは0から始まります。1から始まるわけではないので注意です。

そのため、1番最初のインデックスは「0」、2番目のイン

デックスは「1」、3番目は「2」、4番目は「3」といったように、インデックスはリストが格納されている「順番 - 1」となっています。

リストの作り方

それでは、リストの作り方をみていきましょう。

書き方のきまりは次のとおりです。

変数 = [データ1,データ2,データ3,…]

リストは、角括弧を使って定義をします。

それぞれの要素は、カンマで区切ります。

これだけではわかりにくいと思うので、ソースコードを見ていきましょう。

```
a = ["sato","suzuki","takahashi"]
print(a)
```

このコードは、aという変数

に、’sato’,'suzuki','takahashi'という3つのデータが入ったリストを作るものです。

コードの意味を説明していきます。

最初にリストを代入する変数名を記述します。

イコールを書いて、角括弧を記述します。

これで要素のないリストができました。

次にリストに要素を入れていきます。

それぞれの要素をカンマで区切るのことを忘れないようにしましょう。

これでリストが作成できました。

リストの要素の参照方法

作ったリストの要素にアクセスしてみましょう。

要素にアクセスするにはインデックスを使います。

```
a = ["sato","suzuki","takahashi"]
print(a)
```

順に説明します。

データ型のStringを書いて各括弧。そのあとに配列変数名のarr。イコールを書いて配列に代入したいデータを記述します。

表示させてみましょう。

実行結果：

```
['sato', 'suzuki', 'takahashi']
```

’sato’,'suzuki','takahashi'が表示されました。

次に、配列のデータの変更をしてみましょう。

```
a = ["sato","suzuki","takahashi"]

print(a[0])
```

```
print(a[1])
print(a[2])
```

変数の後に角括弧を付けます。

角括弧の中にインデックスを指定することで、対応したリストの要素にアクセスできます。

実行してみましょう。

実行結果：

```
sato
suzuki
takahashi
```

sato、suzuki、takahashiがそれぞれ表示されました。

リストの要素の変更方法

```
a = ["sato","suzuki","takahashi"]

a[1] = "tanaka"

print(a[0])
print(a[1])
print(a[2])
```

要素を変更してみましょう。

要素を指定して代入をします。

リストの2番目の鈴木さんを田中さんにしてみます。

2番めなので角括弧に書くインデックスは1です。

変更されているか確認してみましょう。

実行結果：

```
sato
tanaka
```

```
takahashi
```

suzukiさんがtanakaさんに変わっています。

多次元リスト

```
a = [["sato","suzuki"],["takahashi", "tanaka
```

```
print(a[0][0])
print(a[0][1])
print(a[1][0])
print(a[1][1])
```

リストの中にリストを入れることもできます。

リストの中にリストを入れる場合は、角括弧の中に角括弧を書き、そこに代入するデータを記述します。

要素へのアクセスについては、角括弧を2回使用すれば良いです。

satoさんは、1つのリストの、1番目です。なので、0の0です。

suzukiさんは、1つのリストの、2番目です。なので、0の1です。

tanakaさんは、2つのリストの、1番目です。なので、1の0です。

takahashiさんは、2つのリストの、2番目です。なので、1の1です。

表示してみましょう。

実行結果：

```
sato
suzuki
takahashi
tanaka
```

satoさん、suzukiさん、takahashiさん、tanakaさんが表示されました。

演算子とは？

演算子とは、足し算、引き算などの四則演算や2つの値の大小を比較するときに使う記号のことです。

算術演算子

算術演算子からみていきましょう。

算術演算子とは、足し算、引き算、掛け算、割り算などをするための演算子です。

早速、ソースコードを見ていきましょう。

```
x = 10
y = 2

print(x + y)
print(x - y)
print(x * y)
print(x / y)
print(x % y)
```

xという変数に「10」、yという変数に「2」を代入。

このxとyの変数を使って、演算子の役割をみていきましょう。

足し算と引き算はそのままプラスとマイナスです。

掛け算はアスタリスク(*)です。

割り算は、スラッシュ(/)です。

最後に剰余です。剰余とは、割り算の余りのことです。

剰余は、パーセント(%)です。

足し算の結果は「12」、引き算は「8」、掛け算は「20」、割り算は「5」、剰余は「0」になるはずです。

実行して、表示させてみましょう。

実行結果：

```
12
8
20
```

```
5.0
0
```

予想通りの結果となりました。

関係演算子

次は関係演算子です。

関係演算子とは、2つの値の関係が正しいか正しくないか判断させる演算子のことです。

例えば、「20」という数字は、「10」より大きいです。これを比較することができます。

関係演算子は、次のレッスンで学ぶ条件分岐のif文などでよく使います。

「10 > 0」という意味は、「10」は「0」より大きいという意味です。

この関係演算子は、正しいです。

2つの関係が正しいときには「True」が返ってきます。

実際に、ソースコードを見ていきましょう。

```
x = 10
y = 2

print(x > y)
```

xに10、yに2を代入します。

x > yが正しいか試してみます。

実行してみます。

実行結果：

```
True
```

Trueが表示されました。

「x < y」としたらどうなるでしょう？

つまり、「10」は「2」より小さいという意味です。

これは誤りです。

```
x = 10
y = 2

print(x < y)
```

正しくないときの「Flase」が返ってくるはずです。

実行してみましょう。

実行結果：

```
False
```

Falseが表示されました。

次に、等価についてみていきましょう。

等価とは、2つの値が等しいということです。

等価はイコールが2つです。

```
y = 2
z = 10

print(x <= y)
print(x >= z)
```

実行結果：

```
False
True
```

等価ではない場合は、エクスクラメーションマークにイコールです。

コードで試してみましょう。

```
x = 10
y = 2

print(x == y)
print(x != y)
```

「10」と「2」は同じではないので、正しくないときの「Flase」が返ってくるはずです。

等価ではないので、こちらは「True」が返ってくるはずで

す。

実行してみましょう。

実行結果：

```
False
True
```

それぞれ「False」と「True」が返ってきました。

論理演算子

次は論理演算子についてみていきましょう。

論理演算子とは、複数の条件を判断させる演算子のことです。

日本語でいうと「かつ」とか「または」のことです。英語でいうとandとorです。

and条件はandを記述し、or条件はorを記述します。

「3」と「8」いう数字でみていきましょう。

例えば、「8」という数字は、5以上かつ10以下です。

「3」という数字は、5以上かつ10以下ではないです。

コードで試してみましょう。

```
x = 8
y = 3

print(x >= 5 and x <= 10)
print(y >= 5 and y <= 10)
```

まずはand条件についてです。

「8」は5以上かつ10以下なので「True」、「3」は5以上かつ10以下ではないので「False」が返ってくるはずです。

実行してみましょう。

実行結果：

```
True
False
```

それぞれ「True」と「False」が返ってきました。

次にor条件についてです。

```
x = 8
y = 3

print(x == 3 or y == 3)
print(x == 1 or y == 1)
```

xは3と等しいか、また、yは3と等しいという条件です。

これはyが3なので、後ろの条件が一致します。

Trueが返ってくるはずです。

xは1と等しいか、yは1と等しいという条件も書いてみまし

ょう。

こちらはどちらの条件にも一致しないので、Falseが返ってくるはずです。

実行してみましょう。

実行結果：

```
True
False
```

予想通りの結果が返ってきました。

代入演算子

次に、代入演算子です。

今まで変数に代入するときに使っていた「=」は代入演算子といいます。

また、代入する時に、足し算や引き算を同時にすることが

できます。

足し算、引き算などと組み合わせて代入する演算子のこと

を複合代入演算子といいます。

例えば、xという変数に「10」を足してから、xに代入する

方法は、「x += 10」です。

例えば、zにyを足してからzに代入する方法は、「z += y」

です。

コードで試してみましょう。

```
x = 8
y = 12
z = 20

x += 10
z += y

print(x)
print(z)
```

xに8、yに12、zに20を代入して、複合代入演算子を試してみましょう。

xは18となり、zは32になるはずです。

実行してみましょう。

実行結果：

```
18
32
```

予想通りの結果が返ってきました。

分岐処理とは？

レッスン4「プログラムの基本構造」で説明しましたが、プログラムの基本的な動きは「順次進行」「条件分岐」「繰り返し」の3つです。

この動画では、「条件分岐」について説明します。

条件分岐とは、条件に合致する場合は「処理A」、そうじゃないときは「処理B」ということができます。

if文

条件分岐の代表例が、if文です。

Pythonでのきまりをみてみましょう

"if 条件:

　条件を満たしたときの処理"

ifに続けて、「条件」を書きます。

コロンの次の行に「条件を満たしたときの処理」を書きます。この行は、インデントをひとつ右にずらして書きます。

実際にコードを書く前に、どのようなコードを書くか説明します。

ageという変数の値が20歳以上なら大人という意味のadultと表示させて、20歳未満なら子供という意味のchildと表示させるといった処理を書きます。

まずは、ageという変数の値が20歳以上なら大人という意味のadultと表示させる処理を書いてみましょう。

```
age = 22

if age >= 20:
    print("adult")
```

年齢という意味のageという変数に「22」という数値を代入しました。

if文の条件にageが20以上と記述します。

22は、20以上です。条件を満たすので、「adult」が表示

されるはずです。

実行してみましょう。

実行結果：

```
adult
```

adultが表示されました。

次に、ageという変数に「18」を代入してみましょう。

条件を満たさないので、何も表示されないはずです。実行してみましょう。

```
age = 18
```

```
if age >= 20:
    print("adult")
```

何も表示されませんでした。

if ~ else文

次にif ~ else文についてみていきましょう。

先ほどみたように、ただのif文は、条件を満たさないとき、

処理されずプログラムが終わります。

一方、if ~ else文は、条件を満たさないときの処理を記述・

実行することができます。

if ~ else文のPythonのきまりをみてみましょう。

先程のif文のあとに、elseと書きます。

elseの次の行もインデントを下げましょう。そのあとに条

件を満たさないときの処理を書きます。

コードを書いてみましょう

```
age = 18

if age >= 20:
    print("adult")
else:
    print("child")
```

ageに代入する数値を18とします。if文の条件は先ほどと同じように、20歳以上とします。

else以降に子供という意味の「child」を表示させる処理を書きます。

18は、20以上の条件を満たさないので、「child」が表示されるはずです。

実行してみましょう。

実行結果：

```
child
```

elseの処理が実行されて「child」が表示されました。

ageに「22」を代入して実行してみましょう。

```
age = 22

if age >= 20:
    print("adult")
else:
    print("child")
```

実行結果：

```
adult
```

今度は条件を満たすので「adult」が表示されました。

if ~ elif ~ else文

もう一つの条件を加えたい場合に使うのが、elifです。

Pythonのきまりをみていきましょう。

先程の条件Aのif文のあとに、elifと書いて2つめの条件Bを書きます。

次の行に、「条件Bを満たしたときの処理」を書きます。この行も、インデントをひとつ右にずらして書きます。

それでは、コードを書いてみましょう

```
age = 0

if age >= 20:
    print("adult")
```

```
elif age == 0:
    print("baby")
else:
    print("child")
```

20歳以上なら「adult」、0歳なら「baby」、それ以外なら「child」というような記述をします。

elifのところに、ageが0だった場合の処理を記述します。

ageの変数に「0」を代入します。

ageは0なので、babyが表示されるはずです。

実行します。

実行結果：

```
baby
```

babyと表示されました。

繰り返しとは？

レッスン4「プログラムの基本構造」で説明しましたが、プログラムの基本的な動きは「順次」「分岐」「繰り返し」の3つです。

繰り返しとは、決まった回数や条件を満たしてれば、同じ処理を実行するプログラム構造です。

for文

繰り返しの代表例がfor文です。

for文は、条件を満たしていれば、同じ処理をぐるぐる繰り返します。

そして、条件を満たさなくなったタイミングで、繰り返しがおわります。

例えば、for文で同じ処理を5回繰り返したい場合で考えてみます。

どうやったら5回をカウントできるでしょうか。

例えば、「1」からスタートして、1ずつ増やしていき「5」

で終われば、5回です。

例えば、「0」からスタートして、1ずつ増やしていき

「4」で終われば、これも5回です。

そういった処理をPythonのfor文で書いていきます。

Pythonでのfor文のきまりをみていきましょう。

for 変数 in range(繰り返し回数):

　繰り返し中に実行する処理

まずforを書きます。次に繰り返し回数を格納する変数を記述します。

ここで定義した変数のことをカウンタ変数といいます。

for文では、繰り返し処理をカウンタ変数によって制御します。

5回繰り返したい場合は、カウンタ変数に数字が順次代入され、5回目がきたら繰り返し処理が終わります。

カウンタ変数は、英語の「index」「Iterator」の頭文字「 i

」が使われることが多いです。

次にinと書いて、range。丸括弧を書きます。

丸括弧の中に繰り返したい回数を、数値で記述します。

最後にコロンを記述します。

コロンの次の行はインデントを下げて、繰り返したい処理を記述します。

繰り返したい処理が終わったら、一番上に戻り、繰り返し処理が終わりになるか判定します。

繰り返し回数分、繰り返していないなら、次のループをします。繰り返し回数分繰り返していたら、ループが終了します。

コードを書いてみましょう。

```
for i in range(5):
    print(i)
```

実行結果：

```
0
1
2
3
4
```

break

次に、break文についてです。

break文は、ある条件にあてはまったとき、繰り返し処理を終了させることができます。

例えば、「0」からスタートさせて、1ずつ増やしていき、「3」になったら繰り返しを終了するといったときです。

コードを書いてみます。

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

ここで、iが3にになったときに、breakする記述をします。

「3」でループを抜けるので、「0」から「2」まで表示されるはずです。

実行します。

実行結果：

```
0
1
2
```

「0」から「2」までが表示されました。

continue

次に、continue文についてです。

continue文は、繰り返し処理で、ある条件にあてはまったときにその処理をスキップしたい場合に使います。

例えば、「0」からスタートさせて、1ずつ増やしていったとき、「3」になったら「3」をスキップさせるという場合です。

コードを書いてみます。

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

if文でiが3にになったときに、処理をスキップさせる

continueを記述をします。

「3」をスキップするので、「0」「1」「2」「4」が表示されるはずです。

実行してみます。

実行結果：

```
0
1
2
4
```

「0」「1」「2」「4」が表示されました。

for文のネスト

for文の中にfor文を入れることもできます。

あるものの中に、それと同じ種類のものが入っている構造のことをネストといいます。

for文の中にfor文が入っている構造のことをfor文のネストといいます。

外側の繰り返しのカウンタ変数は「i」で、0から2まで回し、内側の繰り返しのカウンタ変数は「j」で、0から2まで回すという例で考えてみます。

外側のループの1周目の時に、内側のループが0から2までまわります。内側のループがまわりきったら、外側のループが2周目に入ります。

コードを書いてみましょう。

```
for i in range(3):
    for j in range(3):
        print(i, j, sep="-")
```

iとjの変化がわかるように、「i、ハイフン、j」を表示させてみましょう。

printの第一引数、第二引数をそれぞれi、jにして、sep引数をハイフンにすることで、i、ハイフン、jと表示できます。

iが0周目のときに、jが0から2までまわり、次にiが1周目の動きになる予想通りの結果となりました。

実行結果：

```
0-0
0-1
0-2
1-0
1-1
1-2
2-0
2-1
2-2
```

for文でリスト内を参照

```
arr = [2, 4, 6, 8, 10]
for i in arr:
    print(i)
```

最後に変数を使ってリストの中身を表示させてみましょう。

arrというリスト変数に「2」から「10」までの偶数「2,4,6,8,10」を代入。

for文のinの後にリストを書くことで、リストの中身が変数に一つずつ格納されます。

2から10までの偶数が表示されるはずです。

実行してみましょう。

実行結果：

```
2
4
6
8
10
```

2,4,6,8,10が表示されました。

変数を使って、たし上げていくこともできます。

```
arr = [2, 4, 6, 8, 10]
sum = 0

for i in arr:
    sum += i
print(sum)
```

まず、sumという変数を定義します。

そして、演算子のレッスンで説明した、リストの値を複合代入演算子を使って足し上げていきます。

表示させてみましょう。

実行結果：

```
30
```

足し算ができています。

関数とは？

まず、関数について説明します。

関数とは、いろいろな「処理」をまとめて1つにしたものです。

なぜ関数があるのでしょうか？

料理で例えてみます。

例えば、いつも作るカレーがあるとします。

そのレシピを料理ロボットに記憶してもらいます。

またカレーが食べたくなったときに、

ボタン1つで作れる。

しかも、その料理ロボットは、自分も使えるし、家族も使える。

その料理ロボットが関数なのです。

関数の便利なところは色々あるのですが、3つあげてみます。

1. 同じものを2回書く必要がない

2. 1行で使い回しができる

3. 関数の中のコードを理解していなくても他の人も使うことができる

といった便利な点があります。

関数の種類

関数には2種類あります。

自分で作る関数と、Pythonがあらかじめ用意してくれている関数です。

Pythonがあらかじめ用意してくれている関数のことを「組み込み関数」といいます。

今まで使ってきたprint関数は、組み込み関数です。

print関数はたった1行で変数の中身を表示してくれますが、

print関数の中身は何行ものコードが書かれています。

もしprint関数がなければ、変数の中身を表示させたいときに、イチからそのコードを書くことになり、大変です。

しかし、print関数があることで、コードを書く必要がありませんし、たった1行で使い回すことができます。

また、print関数の中身でどんなことを書かれているか理解していなくても、print関数を使うことができます。

関数の定義、引数、戻り値

def 関数名():

実行する処理

関数を作ることを「関数を定義する」と言います。

Pythonで関数の定義の仕方を見ていきましょう。

Pythonでは関数の定義にdefを使います。

defの後に関数の名前である関数名を書き、

丸括弧を書きます。

丸括弧の中に記述するものを引数と言います。

関数は、引数を受け取ることができます。

受け取った引数は、関数内で使うことができます。

例えば、関数内にある数字と引数を掛け算することができます。

このように、関数に引数の値を渡すことで関数のできる処理の幅が広がります。

引数という言葉は、「引数を関数に渡す」と言ったり、「引数を関数が受け取る」と言ったりします。

引数は、必ず必要と言うわけではなく、省略することができます。

また、引数は、何個でも渡すことができるので、必要な分だけカンマで区切って入れることができます。

引数の丸括弧のあとに

コロンを書いて、

次の行に実行する処理を書きます。

実行する処理の行は右にインデントする必要があります。

関数の記述が

終わったら、

インデントを元に戻します。

そして、関数は、引数を受け取ることができる一方、

関数は処理結果を返すことができます。

これを戻り値といいます。

return文を使うことで、戻り値として関数の外に値を返すことができます。

それでは、実際に、コードを書いてみましょう。

引数なしの関数

それでは、実際に、コードを書いてみましょう。

```
def say_hello():
    print("Hello World")

say_hello()
```

引数なしの関数を見ていきましょう。

文字列の「Hello World」を表示させる関数を定義して実行

してみましょう。

まずは defを書きます。関数名を書きましょう。

関数名は、「say_hello」としましょう。

今回、引数はないので丸括弧のみを書きます。

コロンを書いて、次の行に実行する処理「print("Hello World")」を書きます。

これで関数を定義することができました。

関数を実行するには、関数名、丸括弧で実行できます。

関数名「say_hello」に引数なしの丸括弧。これで「Hello World」が表示されるはずです。

実行してみましょう。

実行結果：

```
Hello World
```

「Hello World」を表示することができました。

```
def say_hello():
    print("Hello World")
say_hello()
say_hello()
say_hello()
```

定義した関数は何度でも呼び出すことができます。

3回say_hello()を記述します。

「Hello World」は3回表示されるはずです。

実行してみましょう。

実行結果：

```
Hello World
Hello World
```

```
Hello World
```

「Hello World」を3回表示することができました。

引数ありの関数

```
def say_hello(greeting):
    print(greeting)

say_hello("Hello World")
```

次に、引数を使う場合の関数を見ていきましょう。

関数の引数を挨拶という意味のgreetingとしましょう。

受け取った引数をprint関数で表示させます。

関数の定義は終わりです。

関数にHello Worldという文字列を渡します。

では、実行してみましょう。

実行結果：

```
Hello World
```

「Hello World」が表示されました。

関数を変数へ代入

```
print(greeting)
hello = say_hello
hello("Good Morning")
```

Pythonは関数を変数に代入することができます。

先ほどと同様に定義した関数名を、そのまま変数に代入す

るだけです。

変数名は「hello」とします。

helloにsay_helloを代入します。

helloのあとに丸括弧、Good Morningの文字列を渡してみましょう。

実行してみます。

実行結果：

```
Good Morning
```

「Good Morning」が表示されました。

複数の引数がある関数

```
def add(num01,num02):
    print(num01 + num02)

add(6,2)
```

引数を2つ使ってみましょう。

足し算という意味の「add」という関数名を作ってみます。

num01とnum02を足してprint関数で表示してみましょう。

では、add関数に6と2を渡してみます。

結果は、5足す2に加えて、関数内の3を足すので、10となるはずです。

実行してみます。

実行結果：

```
8
```

8が表示されました。

return文

```
def add(num01,num02):
    return (num01 + num02)

add(6,2)
```

関数の結果は、return文で返すことができます。

printをreturnに変更してみましょう。

ただし、returnで結果を返しても、printで表示をさせていないので、確認することができません。

念のため、実行してみましょう。

何も表示されません。

```
def add(num01,num02):
    return (num01 + num02)

print(add(6,2))
```

関数を呼び出すaddの部分をprintで括ってみます。

実行してみます。

実行結果：

```
8
```

8が表示されました。

```
def add(num01,num02):
    return (num01 + num02)

add_result = add(6,2)
print(add_result)
```

結果を変数に格納して、print関数で表示させる方法もあります。「add_result」という変数に代入して表示させてみましょう。

表示させてみます。

実行結果：

```
8
```

8が表示されました。

確認問題

それでは最後に確認問題です。

3つの引数を受け取る関数を作り、9と4と2の平均を表示させてください。

一旦、動画を止めて記述してみてください。

答え合わせです。まず、defと書いて、変数名は何でもよいのですが、divという関数を作ります。

引数は3つなので、丸括弧の中に引数を3つ記述します。

そして、returnを書き、a,b,cを足します。足し算は先に計算をしたいので丸括弧で括ってそれから個数の3で割ります。

結果をdiv_resultに格納して、print関数で表示させてみましょう。

実行してみます。

```
def div(a ,b , c):
    return (a + b + c) / 3
```

```
div_result = div(9 ,4 ,2)
print(div_result)
```

実行結果：

```
5.0
```

5.0が表示されました。

クラスとは？

クラスにはインスタンスやコンストラクタなどの概念がでてきます。

私自身、プログラミングを勉強し始めたときに、これを理解するのに時間がかかりました。

私が何冊も書籍を読んで、こういう順番であれば理解しやすいというプロセスで説明します。

最後まで見ていただければ理解できるかと思うので、最後まで見ていってください。

また、最後に確認問題もありますのでぜひ挑戦してみてください。

まずクラスについて説明します。

クラスとは、「データ」と「処理」をまとめたものになります。

Pythonでは、「データ」のことをアトリビュートといい、「処理」のことをメソッドといいます。

アトリビュートとメソッド

アトリビュートは、クラス内で定義された変数のことです。

アトリビュートは、変数と同じように、数値や文字列を代入したり、参照したりすることができます。

クラスにアトリビュートを作ることを「アトリビュートを定義する」と言います。

アトリビュートと変数の違いは、クラスの中にあるかクラスの外にあるかの違いです。

次にメソッドについて説明します。

前のレッスンで関数は、いろいろな「処理」をまとめて1つにしたものと説明しました。

メソッドも関数と同じで、いろいろな「処理」をまとめて1つにしたものです。

簡単にいうと、メソッドは、クラス内に定義された関数です。

メソッドも関数と同じようにdefで定義します。

まとめると、アトリビュートはクラス内の変数、メソッドはクラス内の関数ということになります。

クラスの定義

クラスを作ることをクラスを定義すると言います。

このレッスンでどんなクラスを定義するか説明します。

クラス名はStudentとします。

そのクラスに生徒の名前を代入する「name」というアトリビュートを定義します。

そして、数学と英語の点数の平均を計算するavgというメソッドを定義します。

コードを書いていきましょう

```
class Student:

    def avg():
        print((80 + 70) / 2)
```

まずclassと書いて、次にクラス名を書きます。

今回はStudentというクラス名なので、Student。

クラス名の最初の文字は小文字でも定義はできますが、最初の文字を大文字にするのは、Pythonの慣習となっています。

クラス名の最初の文字は大文字にしましょう。

コロンを書いて改行です。

メソッドの定義

次にメソッドを定義していきます。

数学と英語の点数の平均を計算するメソッドです。

平均を算出するので、平均という意味のaverageを省略して、avgというメソッド名にします。

まずdefと書いて、メソッド名。

丸括弧を書いて、コロン。改行です。

数学が80点、英語が70点を取れたとして、それらを足して2で割ります。

表示させるためにpirnt関数でくくりましょう。
ここまで見た通り、メソッドは関数の定義のやり方と同じです。
ただし、引数について、メソッドと関数に違うところがあります。
スライドで説明します。
メソッドを定義する場合、必ず1つ引数を記述しなければならないです。
関数の場合は、渡したい引数がない場合、空欄でもよいです。
しかし、メソッドの場合は、渡したい引数がない場合でも必ず引数が1つ必要になります。
この引数は、どんな引数名でもよいのですが、selfと書くのが慣習です。
つまり、メソッドに渡したい引数がない場合、メソッドの引数にselfを記述します。
メソッドに渡したい引数が1つの場合、メソッドの引数に

selfと渡したい引数名の合計2つ。

メソッドに渡したい引数が2つの場合、メソッドの引数にselfを含めた合計3つの引数を記述します。

コードを書いていきましょう。

```
class Student:

    def avg(self):
        print((80 + 70) / 2)
```

今回はメソッドに渡す引数がないので、引数の記述は、selfのみです。

このselfの役割は、Pythonがプログラムの実行で使っているものです。

理屈が少し複雑なので、メソッドの引数には、どんな場合でもselfと書くと覚えてしまいましょう。

これでメソッドの定義は終わりです。

クラスを実際に使ってみたいと思いますが、クラスはこのままでは使うことができません。

クラスは、クラスから作られたインスタンスを変数に代入してから使います。

クラスは、インスタンスになって初めて使えるようになります。

コードを書いて、クラスの使い方を見ていきましょう。

```
class Student:

    def avg(self):
        print((80 + 70) / 2)

a001 = Student()
a001.avg()
```

クラスの使い方（インスタンス化）

数学が80点、英語が70点という点数は、aという学級の出席番号001番の人が取ったとします。

変数名をa001とします。

イコールを書いて、クラス名を書き、丸括弧を書きます。

これで、クラスを使えるようになりました。

クラスを使えるような状態にすることを「インスタンス化」「オブジェクト化」「オブジェクト生成」と言ったりします。

インスタンスとは、実体という意味です。

ですから、インスタンス化とは、実体化という意味です。

つまり、インスタンス化とは、クラスという型から、インスタンスという実際に使える「モノ」を作ることを言います。

変数にインスタンスを代入して、インスタンスとして使えるようになったa001は、これからa001インスタンスと呼ぶことにします。

次にメソッドの実行方法についてです。

a001にドットをつけて、メソッド名です。

丸括弧も忘れないでください。

それでは実行してみましょう。

実行結果：

```
75.0
```

平均点の75点が表示されました。

ここまでは、80点と70点を直接、メソッド内に記述していました。

これだと生徒が変わるごとにメソッドの書き換えが必要です。

これを引数で渡して計算できるようにしましょう。

そうすることで、クラスの書き換えは不要になり、クラスを使い回すことができます。

```
class Student:

    def avg(self, math, english):
        print((math + english)/2)

a001 = Student()
a001.avg(80,70)
```

クラス内に記述しているメソッドの2番目の引数をmathとします。

3番目の引数をenglishとします。

そのmathとenglishの引数を、print関数のところに記述します。

avgメソッドに80点と70点を渡して実行してみましょう。

実行結果：

```
75.0
```

75が表示されました。

```
class Student:

    def avg(self, math, english):
        print((math + english)/2)

a001 = Student()
a001.avg(30,70)
```

メソッドに渡す引数を30点と70点にしてみましょう。

平均の50が表示されるはずです。

実行してみましょう。

実行結果：

```
50.0
```

50が表示されました。

アトリビュートの定義

```
class Student:

    def avg(self, math, english):
        print((math + english)/2)

a001 = Student()
a001.avg(80,70)

a001.name = "sato"
print(a001.name)
```

次にアトリビュートについてみていきましょう。

アトリビュートは、クラス内に定義された変数のことです。

a001にドット。アトリビュートを書いて、

値を代入します。

値はsatoさんとしましょう。

これでアトリビュートの定義は終わりです。

print関数で表示させてみましょう。

実行してみましょう。

実行結果：

```
75.0
sato
```

メソッドの結果の75とアトリビュートのsatoが表示されました。

```
class Student:

    def avg(self, math, english):
        print((math + english)/2)

a001 = Student()
a001.avg(80,70)

a001.name = "sato"
print(a001.name)

print(a001.gender)
```

仮に、性別という意味のgenderというまだ定義していない

アトリビュートを表示させてみましょう。

もちろん、定義していないのでエラーになります。

実行してみましょう。

エラーです。

このように未定義のアトリビュートはエラーになります。

```
class Student:

    def avg(self, math, english):
        print((math + english)/2)

a001 = Student()
a001.avg(80,70)

a001.name = "sato"
print(a001.name)

a002 = Student()
print(a002.name)
```

また、a002というインスタンス名でインスタンス化をした

後に、

nameのアトリビュートを表示させてみましょう。

実行してみましょう。

エラーとなりました。

このようにアトリビュートは、インスタンスごとに存在し

ます。

逆の言い方をすれば、インスタンスごとに、アトリビュートを定義しなければなりません。

つまり、インスタンスごとにアトリビュートが存在するので、新しいインスタンスを作るごとに、アトリビュートを定義する必要があります。

そのため、10個インスタンスを作ったとすると、インスタンスごとにアトリビュートを10個定義する記述をしなければなりません。

先ほどの例でいうと、「a001.name」のような記述をインスタンスごとに10個、記述しなければなりません。

その不便さを解消するものがコンストラクタです。

コンストラクタ

```
class Student:

    def __init__(self):
```

```
        self.name = ""

    def avg(self, math, english):
        print((math + english)/2)

a001 = Student()
a001.name = "sato"
print(a001.name)

a002 = Student()
print(a002.name)
```

コンストラクタは、インスタンス化するときに、自動的に実行されるメソッドのことです。

コンストラクタは、初期化メソッドとも言います。

初期化メソッドは、インスタンス化をすれば、必ず実行されるメソッドです。

そのため、後から使うアトリビュートは、初期化メソッドで自動的に作っておけばよいのです。

初期化メソッドの記述方法を見ていきましょう。

初期化メソッドもメソッドです。

メソッドなので、まずdefと記述します。

アンダースコアを2つ。initと書いて、もう一度アンダースコアを2つ。

丸括弧を記述します。

メソッドを定義する場合、最初は必ずselfを書くのでselfを記述。コロンを書いて改行。

これで初期化メソッドの記述は終わりです。

フィールドには、佐藤さん、鈴木さんといったような名前を代入したいので、nameのアトリビュートを定義しましょう。

self、ドット、nameでアトリビュートを定義することができます。

ちなみに、ここでもselfが出てきました。

selfと書くことにより、selfにインスタンスが代入されます。

引数のselfにa001が代入され、self.nameがa001.nameと

なるイメージです。

ここは難しい理屈なので、そういう仕組みになっているのだと思って覚えておきましょう。

ここでは、ダブルクオテーション2つで、空の値を代入させておきましょう。

では、インスタンス化をして、a001とa002のnameの中を見てみましょう。

avgメソッドの記述は消しておきます。

a001にはsatoが、先ほどエラーになったa002には初期化メソッドでアトリビュートを作ったので、

エラーにならず、空の値が入っているはずです。

実行してみましょう。

実行結果：

```
sato
```

エラーにならずに、satoと空の値が表示されました。

```
class Student:

    def __init__(self):
        self.name = ""

    def avg(self, math, english):
        print((math + english)/2)

a001 = Student()
a001.name = "sato"
display(print(a001.name))

a002 = Student()
a002.name = "tanaka"
display(print(a002.name))
```

a001にsatoを代入してみましょう。

a002にtanakaを代入してみましょう。

実行してみます。

実行結果：

```
sato
tanaka
```

satoとtanakaが表示されました。

```
class Student:

    def __init__(self,name):
        self.name = name
```

```
    def avg(self, math, english):
        print((math + english)/2)

a001 = Student("sato")
print(a001.name)

a002 = Student("tanaka")
print(a002.name)
```

アトリビュートは、インスタンス化と同時に代入することもできます。
初期化メソッドの第2引数にnameという引数を記述します。
ダブルクォーテーションで空を代入していたところに、nameを記述します。
イメージとしては、第2引数のnameを初期化メソッド内のnameが受けて、それをself.nameに代入します。
では、a001にインスタンス化と同時に"sato"を渡してみましょう。
a002インスタンスにも"tanaka"を渡します。
表示させてみましょう。

実行結果：

```
sato
tanaka
```

satoさんとtanakaさんが表示されました。

クラスの便利なところ

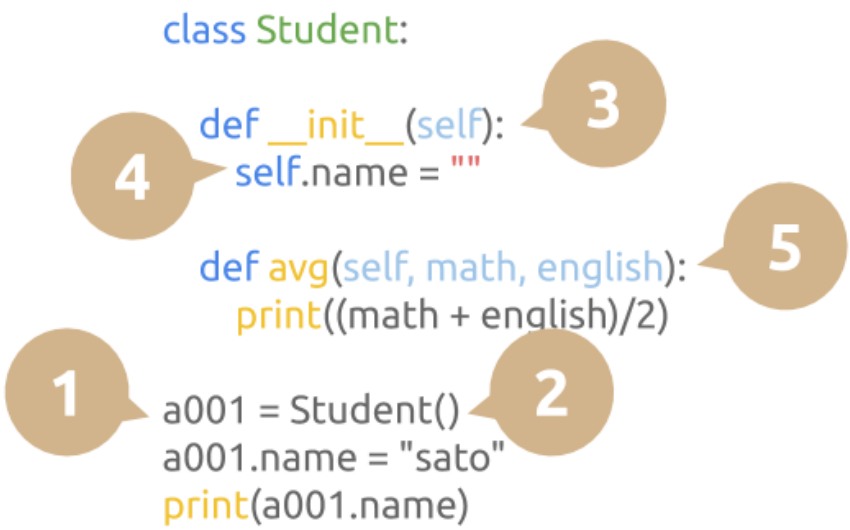
以上がクラスの使い方です。
最後に、クラスの便利なところはどんなところでしょう？
クラスは1度、定義しておけば、後からいくらでもインスタンスを作ることができます。
車を作る「設計図」がクラス、たい焼きを作る「金型」がクラスと表現したりもします。
もう1つ表現するなら、クラスは、パソコンで使うコピペです。
コピペでどんどんインスタンスを作ることができます。

もしクラスがなければ、生徒ひとりひとりのためにStudentクラスを書かなければいけないので面倒です。
クラスがあるから効率よくプログラミングすることができます。

確認問題

最後に確認問題をやっていきましょう。
このレッスンでは新しい用語がたくさん出てきたので、確認してみましょう。

① a001のことは何と言うでしょうか？
② Student()は何を呼び出しているのでしょうか？
③ def **init**(self):の部分は何と言うでしょうか？
④ sef.nameは何を定義しているのでしょうか？
⑤ def avg(self, math, english):は何と言うでしょうか？



1がインスタンス

2がクラス

3がコンストラクタ、初期化メソッドとも言います。

4はアトリビュート

5はメソッドです。

実践

総まとめの実践編として、テスト結果を判定するプログラムを作ってみましょう。

どんなプログラムを書くか説明します。

まず、Studentというクラスを作ります。

Studentクラスには、生徒の名前を代入する「name」のアトリビュートを定義します。

そして、Studentクラスには2つメソッドを定義します。

1つのメソッドは、5教科の平均点を計算するcalculate_avgメソッド。

ちなみに、calculateは計算するという意味で、Avgは、平均のaverageを省略する時にプログラミングでよく使う文字です。

もう1つメソッドは、平均点以上だったら合格という意味の"passed"を表示、平均点以下なら不合格という意味

の"failed"を表示させるjedgeメソッドを定義します。

早速、ソースコードをみていきましょう。

```
class Student:

    def __init__(self,name):
        self.name = name

    def calculateAvg(self, data):
        sum = 0
        for num in data:
            sum += num

        avg = sum / len(data)
        return avg

    def jedge(self, avg):

        if (avg >= 60):
            result = "passed"
        else:
            result = "failed"
        return result
```

```
a001 = Student("sato")
data = [70, 65, 50, 90, 30]
avg = a001.calculateAvg(data)
result = a001.jedge(avg)

print(avg)
print(a001.name + " " + result)
```

まずClassと書いて、次にクラス名を書きます。

今回はStudentというクラス名なので、Student。コロンを書いて改行。

次に、コンストラクタ（初期化メソッド）の定義していきましょう。

佐藤さん、鈴木さん、佐々木さんといったような名前を代入したいので、アトリビュートはnameとしておきましょう。

次に、5教科の平均点を計算するcalculate_avgメソッドを定義します。

メソッドには、5教科の点数が格納されているリストを渡し

ます。
そして、sumという変数を定義をして、0を代入。
レッスン11で説明しましたが、for文のinの後にリストを書くことで、リストの中身が変数に一つずつ格納されます。
それをレッスン09の演算子で説明した、配列の値を複合代入演算子を使って足し上げていきます。
そして、合計を、リストの要素数で割って平均を算出し、avgという変数に代入します。
リストの要素数を求めるにはlenを使います。
算出した値をreturnで返します。
最後に、テスト結果を判定するjedgeメソッドを作ってみましょう。
jedgeメソッドに平均点を渡しましょう。引数名はavgとします。
if文で、その平均点が60点以上ならpassed、それ以外ならfailedが、resultという変数に格納されるようにします。
これもreturnで返します。

これで、jedgeメソッドの完成です。

以上で、アトリビュートとメソッドの定義は完了です。

続いて、「インスタンス化」をします。

aという学級の出席番号001番の人は、satoさんだとします。

したがって、インスタンス名をa001とします。

「sato」を渡してインスタンス化します。

そして、dataという変数に、リストを代入。リストには、70, 65, 50, 90, 30という5科目の点数を記述します。

calculate_avgメソッドにリストを渡して平均点を算出します。

returnで平均点が返ってくるので、それをavgという変数に格納します。

そのavgをjedgeメソッドに渡すと、passedかfailedが返ってくるはずです。

print関数で、平均点を表示させ、名前と結果を表示させましょう。

実行してみましょう。

61点という平均点と名前とpassedが返ってきました。

実行結果：

```
61.0
sato passed
```

次にリストの90点を10点にしてみましょう。

```
class Student:

    def __init__(self,name):
        self.name = name

    def calculateAvg(self, data):
        sum = 0
        for num in data:
            sum += num
```

```
        avg = sum / len(data)
        return avg

    def jedge(self, avg):

        if (avg >= 60):
            result = "passed"
        else:
            result = "failed"
        return result

a001 = Student("sato")
data = [70, 65, 50, 10, 30]
avg = a001.calculateAvg(data)
result = a001.jedge(avg)

print(avg)
print(a001.name + " " + result)
```

45点になるので、satoとfailedが表示されるはずです。

実行してみます。

45点と名前とfailedが返ってきました。

実行結果：

```
45.0
sato failed
```

これでpythonの超入門コースはおわりです。お疲れ様です。