# Python基礎

基本文法編

# Python基礎

目次

- 変数と標準入出力
- 条件分岐と比較演算子
- 繰り返し文
- 関数定義の基本
- データ構造と内包表記
- クラスとオブジェクト指向
- 継承、ポリモーフィズム

# Python基礎

講義資料と講義用colabについて

本講義資料は、鬼木の許可なしに第三者に共有することを禁止しております。

万が一共有する場合は、事前に連絡頂きますようお願いいたします。

また、講義で扱った単元のサンプルコードを共有しています。

下記リンクからアクセスできますので、ご活用ください！

https://colab.research.google.com/drive/16kpZTlLKFj69OZ9trNq7trjr1uT-up0c?usp=sharing

# 変数と標準入出力

```
print("あいうえお")
print(10)
print(15e4)
print(10e-3)
```

- Pythonで文字や数字を出力する場合は、print文を使うことが多いです。
- 巨大な数値や小数を扱う場合、桁数が増えると、0を打つのが面倒になることがあります。その際、”e”という文字を使うことで、10の何条かを表せる方法もあるので覚えておきましょう。

出力結果

```
soshioniki@SoshiOnikinonotobukkukonpyuta python % python print.py
あいうえお
10
150000.0
0.01
```

# 変数と標準入出力

```
# 変数
a = 10 #int型
b = 3.1415 #float型
c = "おにぎり" #string型
d = "10" #string型
e = True #bool型
```

- Pythonの変数定義は、C言語などと違って（変数の）型宣言が不要。
- 左図に示したものが代表的な変数の型。

# 条件分岐と比較演算子

```
x = int(input("数字を入力："))

if x > 5:
  print("x は 5 より大きい")
elif x == 5:
  print("x は 5 です")
else:
  print("x は 5 より小さい")
```

- Pythonにおける基本的な条件分岐の書き方は左図に示した通り。
- 左の命令は、ユーザーに数値を入力してもらう場合のものだが、最初から変数の中に数値が入っていても問題ない。

# 条件分岐と比較演算子

| 記号 | 意味 |
| --- | --- |
| A==B | AとBは等しい |
| A!=B | AとBは等しくない |
| A&&B | AかつB |
| A\|\|B | AまたはB |
| A>=B | AはB以上 |
| A<=B | AはB以下（未満） |
| A＞B | AはBより大きい |
| A<B | AはBより小さい |
| A/B | AをBで割る（商を出す） |
| A%B | AをBで割ったあまり |

- Pythonにおける基本的な比較演算子は左図に示した通り。
- 基本的にはCやC++と変わらない。

# 繰り返し文-for文

```
for i in range(10): #終了値のみ
print(i)
print("_________")
for x in range(1,5):#開始値と終了値のみ
print(x)
print("_________")
for y in range(1,5,2):#開始値と終了値、
間隔指定
print(y)
```

- Pythonにおける基本的な繰り返し文（for文）の書き方は左図に示した通り。

for文の書き方として、

① 終了値のみ指定

② 開始値と終了値を指定

③ 開始値、終了値と間隔を指定

の3パターンある。

# 繰り返し文-while文

```
a = 0
while a <5:
  a+=1
  print(a)
```

- Pythonにおける基本的な繰り返し文（while文）の書き方は左図に示した通り。
- For文とwhile文は、目的によって使い分ける必要がある。

| 構文 | 用途 |
| --- | --- |
| For文 | あらかじめ反復回数が決まっている時 |
| While文 | 繰り返しの終了条件を制御したい時 |

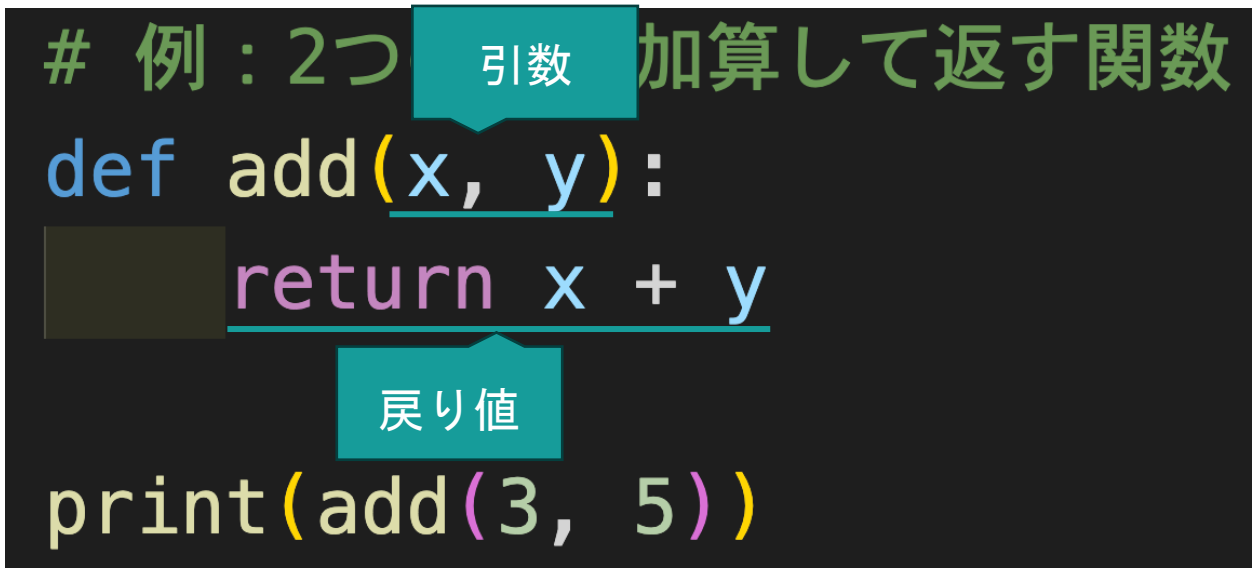
# 関数定義の基本

```python
# 例：2つの数を加算して返す関数
def add(x, y):
    return x + y

print(add(3, 5))
```

- 関数定義の基礎を見ていこう。
- 関数定義をするうえで大切な用語を下にまとめる。

| 用語 | 意味 |
| --- | --- |
| 引数 | 関数に渡す入力データのこと。 |
| 戻り値 | 関数が呼び出し元に返す結果データ のこと。通常、return 文で指定する。 |

# 関数定義の基本




- 関数定義の基礎を見ていこう。
- 関数定義をするうえで大切な用語を下にまとめる。

| 用語 | 意味 |
| --- | --- |
| 引数 | 関数に渡す入力データのこと。 |
| 戻り値 | 関数が呼び出し元に返す結果データ のこと。通常、return 文で指定する。 |

# データ構造と内包表記-リスト

```python
a = [1,2,3,4,5,6,7,8,9]
#インデックスアクセス（添字）
print(a[0])
print(a[-1])
# スライス
print(a[1:4])
print(a[:3])
print(a[::2])
```

- Pythonにおけるリストの使い方は左図の通りです。
- リストへのアクセスは、

① 添字を使ってアクセスする。

② スライスでアクセス（複数範囲指定可能）

の２つがある。

- 出力結果は下の図のとおり。

```
soshioniki@SoshiOnikinonotobukkukonpyuta python % python print.py
1
9
[2, 3, 4]
[1, 2, 3]
[1, 3, 5, 7, 9]
```

# データ構造と内包表記-リスト

```
1 a = [1,2,3,4,5,6,7,8,9]
2 #インデックスアクセス（添字）
3 print(a[0])
4 print(a[-1])
5 # スライス
6 print(a[1:4])
7 print(a[:3])
8 print(a[::2])
```

【解説】

- リストの要素にアクセスする時は、

リスト名[添字]

と書くか、スライスを使う。

- 6~8行目のスライスでの書き方について、

6行目: 1番目から4番目までを出力

7行目: 0番目から3番目までを出力

8行目: 0~8番目までの中で、1個おきに要素を出力。

# データ構造と内包表記-リスト

```
lst = [1, 2, 3]

lst.append(4) # 末尾に追加
lst.insert(1, 9) # 最初に追加
lst.extend([5, 6]) # 末尾に追加
print(lst)

lst.remove(2) # 2を削除
p = lst.pop() # 末尾を取り出し
q = lst.pop(1) #添字の1番目を取り出し
print(lst)
```

実行結果

```
soshioniki@SoshiOnikinonotobukkukonpyuta python % python print.py
[1, 9, 2, 3, 4, 5, 6]
[1, 3, 4, 5]
```

続いて、リストの要素の操作について概観する。

ここではわかりやすく、表でまとめる。

| 操作、命令 | 意味 |
|---|---|
| .append | 要素をリストの末尾に追加 |
| .insert | 要素を最初に追加 |
| .extend | 要素を末尾に追加 |
| .remove | 指定した要素を削除 |
| .pop | リストの末尾の要素を取り出す |

# データ構造と内包表記-辞書型

```python
prices = {'pen': 10, 'notebook': 30}

# 通常アクセス（キーがないと KeyError）
print(prices['pen'])

# get を使ってデフォルト値を返す
print(prices.get('eraser'))
print(prices.get('eraser', 0))

# in 演算子で存在確認
if 'pen' in prices:
    print('ペンがあります')
```

続いて、辞書型の要素のついて説明する。

辞書型はリストと違い、キーと値をペアで保持することができることが大きな特徴である。

ここでは、キーや値にアクセスする方法を示す。

実行結果

```
soshioniki@SoshiOnikinonotobukkukonpyuta python % python print.py
10
None
0
ペンがあります
```

# データ構造と内包表記-辞書型

```
#空のリストを作成
d = {}

# 代入で追加
d['a'] = 2
print(d)

# update:キーと値を一括追加（更新）
d.update({'b': 3, 'c': 4})
d.update(d=5, e=6)
print(d)
```

続いて、辞書型の要素の操作について。

ここではわかりやすく、表でまとめる。

| 操作、命令 | 意味 |
| --- | --- |
| .update | キーと値を一括追加（更新） |
| .pop | 指定したキーと値を削除 |
| .popitem | （最後に追加された）要素をタプル型で取り出す |

# データ構造と内包表記

```python
#リストとfor文
a = [10,11,12,13,14,15]
for i in a:
    print(i)

#辞書型とfor文
b = {1:"apple",2:"banana",3:"orange"}
for number,fruit in b.items(): #キーと値をセットで取得
    print(f"{number},{fruit}")
```

また、リストや辞書型は、for文と相性が良いことも抑えておこう！

リストや辞書型をfor文で使うと、キーや値を取り出すことが可能となる。

実行結果

```
soshioniki@SoshiOnikinonotobukkukonpyuta python % python print.py
10
11
12
13
14
15
1,apple
2,banana
3,orange
```

# データ構造と内包表記

```python
#辞書型とfor文
b = {1:"apple",2:"banana",3:"orange"}
for key,fruit in b.items():
    print(f"{key},{fruit}")
print("_________")
c = {1:"apple",2:"banana",3:"orange"}
for key in c.keys():
    print(key)
print("_________")
d = {1:"apple",2:"banana",3:"orange"}
for fruit in c.values():
    print(fruit)
```

ここで、辞書型の操作について表にまとめる。

| 命令 | 意味 |
| --- | --- |
| .items() | 辞書内のキーと値をセットで取得できる。 |
| .values() | 値のみを取得 |
| .keys() | キーのみ取得 |

実行結果

```
soshioniki@SoshiOnikinonotobukkukonpyuta python % python print.py
1,apple
2,banana
3,orange
_________
1
2
3
_________
apple
banana
orange
```

# データ構造と内包表記

```python
# 例：1〜5 の平方をリストとして作成
squares = [n**2 for n in range(1, 6)]
print(squares)

# if 条件を加える
even_squares = [n**2 for n in range(1, 11) if n % 2 == 0]
print(even_squares)
```

続いて、for文を簡潔に書くことができる「内包表記」について説明する。

内包表記の書き方は以下の通り。

[評価式 for 変数 in イテラブル]

左図に具体例を示す。

実行結果

```
soshioniki@SoshiOnikinonotobukkukonpyuta python % python print.py
[1, 4, 9, 16, 25]
[4, 16, 36, 64, 100]
```

# データ構造と内包表記
# -辞書内包表記-

また、辞書型についても「内包表記」で処理を書くことができる。

内包表記の書き方は以下の通り。

{キー式: 値式 for 変数 in イテラブル}

左図に具体例を示す。

```python
# 例：1〜5 の数をキー、その平方を値とした dict
square_dict = {n: n**2 for n in range(1, 6)}
print(square_dict)

# if 条件付き
odd_dict = {n: n**2 for n in range(1, 11) if n % 2 == 1}
print(odd_dict)
```

実行結果

```
soshioniki@SoshiOnikinonotobukkukonpyuta python % python print.py
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
{1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
```

# クラスとオブジェクト指向

クラスについて説明していきます。

よく、クラスは設計図に例えて説明されます。たとえば、車を製造しよう とした時に必要なものは、タイヤ、エンジン、バッテリー、速度計、座席 …など、たくさんのものが必要になってきます。

クラスは設計図なので、「モノ」ではなく、**「どんなものを作るかを定義 する枠組み」**を指します。 そして、クラスから作られる具体的な「モノ」を**インスタンス**と言います。

# クラスとオブジェクト指向

先ほど述べた通り、 クラスはあくまで設計図に過ぎないので、実際に作って動かしてみないといけないわけです。

この時、クラスという設計図をもとに、具体的なオブジェクト（インスタンス）を作ることを**「インスタンス化」**と呼びます。
また、クラスを作る時には、関数定義が多く使われます。この、クラス の中で定義される関数のことを**「メソッド」**と呼びます。
また、インスタンスを作る過程で、一度だけ実行される特別なメソッド を、**「コンストラクタ」**と呼びます。

# クラスとオブジェクト指向

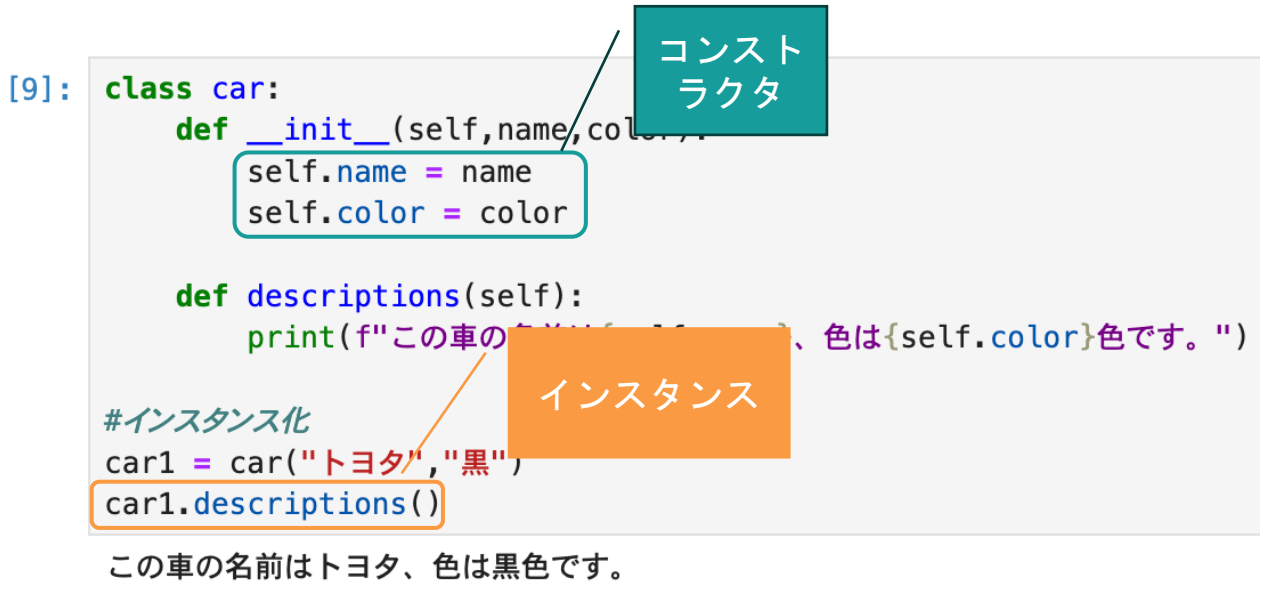
[9]:
```
class car:
    def __init__(self,name,color):
        self.name = name
        self.color = color

    def descriptions(self):
        print(f"この車の名前は{self.name}、色は{self.color}色です。")

#インスタンス化
car1 = car("トヨタ","黒")
car1.descriptions()
```

この車の名前はトヨタ、色は黒色です。

左図が、クラスの実装例となる。

このコードが、クラス設計のベースとなる。

# クラスとオブジェクト指向




「クラス」「インスタンス」「コンストラクタ」などの用語をしっかりと抑えておこう！

# クラスとオブジェクト指向

クラスのインスタンス自身を指す
引数。おまじない（慣習）と考える。

[9]:
```
class car:
    def __init__(self,name,color):
        self.name = name
        self.color = color

    def descriptions(self):
        print(f"この車の名前は{self.name}、色は{self.color}色です。")

#インスタンス化
car1 = car("トヨタ","黒")
car1.descriptions()
```

```
この車の名前はトヨタ、色は黒色です。
```

コード解説

# クラスとオブジェクト指向

[9]:
```python
class Dog:
    def __init__(name):  # self がない
        name = name  # 正しく初期化されない

    def bark():
        print(f"{name}が吠えています！")  # エラーが発生する

dog1 = Dog("Pochi")  # エラー発生
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[9], line 8
      5     def bark():
      6         print(f"{name}が吠えています！")  # エラーが発生する
----> 8 dog1 = Dog("Pochi")  # エラー発生

TypeError: Dog.__init__() takes 1 positional argument but 2 were given
```

## 【Tips】

クラスを定義する際、selfを引数に置かないとエラーになります！

# クラスの継承、ポリモーフィズム

```python
#継承
class animal:
    def __init__(self,name):
        self.name = name

class Dog(animal):
    def sound(self):
        print(f"{self.name} saysワンワン")

animal = [Dog("太郎"),Dog("ポチ")]

for a in animal:
    a.sound()
```

```
太郎 saysワンワン
ポチ saysワンワン
```

ここからはクラスの応用となる「継承」と「ポリモーフィズム（多態性）」について扱います。

それぞれの用語の意味を抑えましょう！

継承：子クラスが親クラスの属性やメソッドを引き継ぐ

多態性（ポリモーフィズム）：同じメソッド名で異なる動作を持つ（実行時に適切なメソッドが呼ばれる）

# クラスの継承、ポリモーフィズム

```python
#継承
class animal:
    def __init__(self,name):
        self.na
class Dog(animal):
    def sound(self):
        print(f"{self.name} saysワンワン")

animal = [Dog("太郎"),Dog("ポチ")]

for a in animal:
    a.sound()
```

親クラス

引数に親クラスを持つ（継承）

子クラス

リストでインスタンスを生成

各要素に対して、soundメソットを呼び出す

```
太郎 saysワンワン
ポチ saysワンワン
```

左図のコードは、クラスの継承のサンプルコードです。

解説は左図の通り。

# クラスの継承、ポリモーフィズム

```python
#ポリモーフィズム（多態性）の実装例
# 犬を表すクラス定義
class Dog:
    def __init__(self, name):
        self.name = name

    # speak メソッド：犬の鳴き声を返す
    def speak(self):
        return f"{self.name} says ワン！"

# 猫を表すクラス定義
class Cat:
    def __init__(self, name):
        self.name = name

    # speak メソッド：猫の鳴き声を返す
    def speak(self):
        return f"{self.name} says ニャー！"

# Dog と Cat のインスタンスをまとめてリストに格納
animals = [Dog("ポチ"), Cat("タマ")]

# リスト内の各オブジェクトに対して speak() を呼び出し、多態性（Polymorphism）を示す
for a in animals:
    print(a.speak())
    # Dog インスタンスなら Dog.speak が、
    # Cat インスタンスなら Cat.speak が呼ばれる
```

左図のコードは、ポリモーフィズム（多態性）のサンプルコードです。

解説は左図の通り。

犬と猫で、出力結果が変わることが確認できればOK！

# Python基礎 終了！

ここまでで、Pythonの基礎部分の学習は終了しました！

お疲れ様でした！

理解できていない部分も多いかもしれませんが、今後触れていくことで慣れると思います！

良ければ中級向けの資料も見てみてくださいね！