第 2 回 Python 講座

2018 年 7 月 9 日

1　制御構文

1.1　while

python で処理を繰り返す場合 *while* 文と *for* 文の二つがあります。まず *while* 文ですが C 言語とほぼ変わらないので構文だけ紹介して実際のプログラムソースは割愛させてもらいます

while

```
while 条件:
    処理
```

なお python では C 言語のように {} で囲むのではなくインデントで管理されていることに気を付けてください。

1.2　for

python で最も使う繰り返し構文が *for* 文です。まず *for* 文における構文を下に示します

for

```
for ループ変数 in 範囲:
    処理
```

ここで範囲に使うの *range*(*a*) で、この a には範囲が入ります。range には主に 3 種類あって。range の中に入る数字の数で種類分けされます。

まずは下のプログラムを実行してください。

例 1

```
v=0
for i in range(10):
    v=v+i
    print(v)
```

この実行結果を見ると 0 から 9 までの数字を足していっています。*range* の中の数字が 1 つだとそれは回数を指定することになります。よってこれは 0 から 9 までの十回を繰り返し足すプログラムとなっています。

次にこのプログラムを実行してください。

例 2

```
v=0
for i in range(1,11):
    v=v+i
    print(v)
```

この実行結果を見ると 1 から 10 までの数字を足していっています。range の中の数字が 2 つになると 1 つ目が開始値、2 つ目が終了値を指定しています。この時の終了値では実際に数字の-1 までを実行します。

では最後にこのプログラムを実行してください

例 3

```
v=0
for i in range(1,21,2):
    v=v+i
    print(v)
```

この実行結果を見ると 1 から奇数だけを足していっています。1 つ目 2 つ目は先ほどと変わらず開始値終了値を指定していて、3 つ目がステップ値を指定しています。ステップ値とは通常 1 ずつ上がっていくループ変数の値の変化を変えることが出来ます。今回は 2 を指定したため 1 個飛ばしで足していくプログラムとになっています。

1.3 制御構文で使う他の処理

python でも continue 文及び break 文を使うことが出来ます。この使い方も C 言語と変わらないため割愛させてもらいます。

2 リスト

2.1 リストの基本操作

では次にリスト操作をやっていきます。まずリストの宣言方法は以下のとおりです。

リストの生成

```
変数=[数, 数]
```

なおこの数の部分は文字列でも構いません。後、文字列の扱いについては次項で詳しく触れます。このリストの生成では各括弧を間違えてはいけません。なぜなら通常の括弧 () や中括弧 {} ではリストではなくタプルや集合型として扱われてしまうためです。

リストの要素にアクセスしたりリストの中身を更新する方法はリストの要素数を宣言して演算子を使用することで可能になります。

基本操作

```
a=[0,1,2,3,4,5]
print(a[0])
a[0]=6
print[a]
```

コマンドライン

```
0
[6,1,2,3,4,5]
```

ここで便利な関数として $len()$ 関数というものがあります。$len()$ 関数ではリストの要素数を調べることが出来ます。構文としては len(リスト名) で指定したリストの要素数を調べることが出来ます。例えば要素数が 10 のリストでならば $len()$ を用いると 10 と返ってきます。

2.2 for とリストの組み合わせ

for 文とリストを組み合わせることでリストの中身を一気に処理することが出来ます。

では次のプログラムを実行してください。

例 4

```
v=[76,88,93,27,60,100]
sum_v=0
for i in v:
    sum_v=sum_v+i;
    print(sum_v)
```

このプログラムはリストの中身を合計するプログラムとなっています。これはループ変数 i がリストの要素にアクセスしてそれを足していってるため行うことが出来ます。このように for 文とリストを組み合わせることにより、簡単にリストの中身を処理することが出来ます。

2.3 リストで便利な関数、メソッド

python のリスト操作では便利な関数、メソッドが多いです。その中でも使うことの多いもののプログラム例を示して、残りを表で示すこととします。

例えば先ほどリストの中身の合計を計算するプログラムを実行してもらいましたが、リストの中身の合計を計算する関数はもともと存在します。それが $sum()$ 関数です。

では先ほどのプログラムを書き換えてください。

sum

```
v=[76,88,93,27,60,100]
sum_v=sum(v)
print(sum_v)
```

このように書き換えると計算途中を表示することはできないが計算結果だけを表示することが出来ます。この

ように計算家庭がいらないのならば $sum()$ 関数を使う方が for 文を使わなくてもよいため見やすいプログラムと言えると考えます。

次に *append* メソッドを試してください。これは現在あるリストの末尾に新たな要素を追加するメソッドです。

append メソッド

```
v=[76,88,93,27,60,100]
print(v)
v.append(15)
print(v)
```

このようにメソッドなので. の後につけて使います。

下に示すのは関数、メソッドの種類です。この後の演習にてこの関数、メソッドは使ってもらいます。

メソッド	機能の説明
$append(x)$	値 x をリストの末尾に追加する
$extend(L)$	異なるリスト L を末尾に追加する
$insert(i,x)$	インデックス i の位置に x を挿入する
$remove(x)$	リストにある値 x を削除する。ただし最初の 1 つのみ
$pop()$	リストの末尾にある要素を取り出しリストから削除する
$clear()$	リストの全要素を削除
$index(x)$	リストから x を探してその位置 (インデックス) を返す
$count(x)$	リスト中に値 x が何回出現するか回数を返す
$sort()$	リストの中身を昇順に並び替える
$reverse()$	リストの中身を降順に並び替える
$copy()$	リストの複製を返す

関数	機能の説明
$sum(L)$	リスト L の合計を返す
$max(L)$	リスト L の最大値を返す
$min(L)$	リスト L の最小値を返す

またこれらとは違い、要素を削除する *del* が存在します。

del

```
del リスト [要素数]
```

これで特定の要素を削除することが出来ます。

2.4　スライス

python にはスライスと呼ばれる機能が存在する。これはリストの一定範囲の要素に対して操作することが出来る。

スライス

```
v=[0,1,2,3,4,5,6,7,8,9]
print(v[2:4])
print(v[:4])
print(c[2:])
```

コマンドライン

```
[2,3]
[0,1,2,3]
[2,3,4,5,6,7,8,9]
```

このように *print* 文で表示させたり、先ほど紹介した *del* コマンドを使うことが出来ます。

3　文字列

3.1　基本操作

python では文字列に対して様々な操作を行うことが出来ます。また文字列を扱う際にはリストとの関係を整理することにとってさらに使いやすくなります。

まず文字列を生成は ” または ′ で囲むことにより生成することが出来ます。

またこの文字列に対して演算子により演算を行うことが出来ます。では以下のプログラムを実行してください。

生成、演算

```
s1=”abcdefg”
s2=’hijklmn’
print(s1)
print(s2)
s3=s1+s2
print(s3)
s4=s1*3
print(s4)
```

コマンドライン

```
abcdefg
hijklmn
abcdefghijklmn
abcdefgabcdefgabcdefg
```

python で文字列は宣言したときにリストに格納されます。先ほどのプログラムに $print(s1[1])$ を加えてください。そうすると s1 の 1 番目の要素である b が表示されます。またこの文字列でもスライスを行うことが出来ます。宣言方法は先ほどリストでやったとおりです。

3.2 文字列のメソッド

文字列でも便利なメソッドが多数あります。例えば文字列を分割する $split()$ メソッドや結合する $join()$ メソッドがそれにあたります。

それでは次のプログラムを実行してください

メソッド

```
str=”This is a pen”
print(str)
str2=str.split()
print(str2)
str3=”-”.join(str)
print(str3)
```

コマンドライン

```
This is a pen
”This”,”is”,”a”,”pen”
This-is-a-pen
```

このように $split()$ メソッドでは文字列を空白で区別することが出来て、$join()$ メソッドにより-でつなげています。また $split()$ メソッドでは他の文字で区切ることもできます。

この次のページにに文字列で使えるメソッドを紹介します。

メソッド	機能の説明
$str.find(a)$	str のどこに a があるか調べる。ない場合は-1 を返す
$str.lower$	文字列 str を小文字にする
$str.upper$	文字列 str を大文字にする
$str.format(v1)$	文字列 str に値 $v1$ を書式化して埋め込む
$str.replace(a,b)$	文字列 str の部分文字列 a を b に変える
$str.split(a)$	文字列を a で分割する
$a.join(str)$	str を a でで結合して返す
$str.startswith(a)$	str が a で始まるなら True を返す
$str.isnumeric()$	文字列がすべて数字ならば True を返す

3.3 文字列の扱い

$print()$ 関数上にて文字列及び変数の値を同時に表示する際には主に 3 つのパターンがあります。

p

```
rint(”文字列”+ 変数 +”文字列”)
print(”文字列”, 変数”文字列”)
print(””文字列 10 文字列 21”.format(変数 1, 変数 2))
```

この 3 パターンなら基本的に書くことはできますが微妙に表示が異なります。また 1 つ目の＋を使うパターンでは数値のままでは出力することが出来ません。この場合一度数値を文字列に為す必要があります。それでは次のプログラムを実際に実行してみてください。また 3 個目の $format$ メソッドは上の表にも記載していますのでわからなくなったらこちらを参照してください。

文字列操作

```
a=10
b=15
a_m=str(a)
b_m=str(b)
print(”a=”+a_m+”,b=”+b_m)
print(”a=”,a,”,b=”,b)
print(”a=0,b=1”.format(a,b))
```

コマンドライン

```
a=10,b=15
a= 10 b= 15
a=10,b=15
```

4　関数定義

python では長いプログラム等を書く際にはプログラムをより簡潔にするために関数を使います。python における関数の定義は次の方法で行います。

関数定義

```
def　関数名 (引数):
    処理
    return
```

では実際に関数で掛け算を行うプログラムを書きましょう。次のプログラムを実行してください。

関数

```
def cal(a,b):
    "' 掛け算を行う関数"'
    return a*b
print(cal(2,3))
print(cal(5,6))
```

今定義した関数の中に三つの' で囲まれた範囲があります。これは *docstring* と言って関数の説明をするときに使います。それでは先ほどのプログラムの最後に help(cal) を追加して再度実行してみてください。

また python で関数を定義するときには引数の値を正確に決めなくても行うことが出来ます。それが可変長引数です。

可変長引数

```
def sumargs(*args):
    v=0
    for n in args:
        v+=n
    return v
print(sumargs(1,2,3,4))
print(sumargs(10,46,38,6,29,88))
```

コマンドライン

```
10
217
```