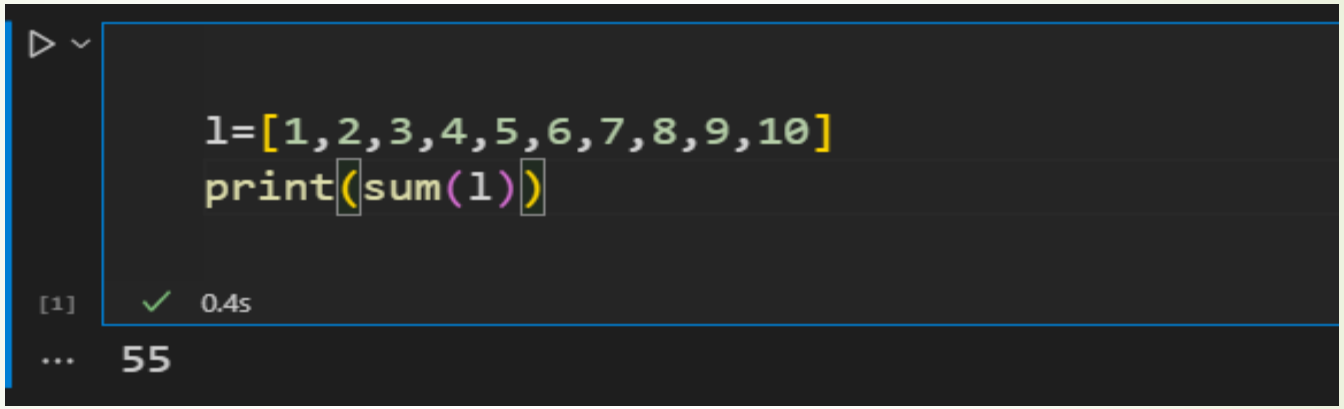
python輪読会

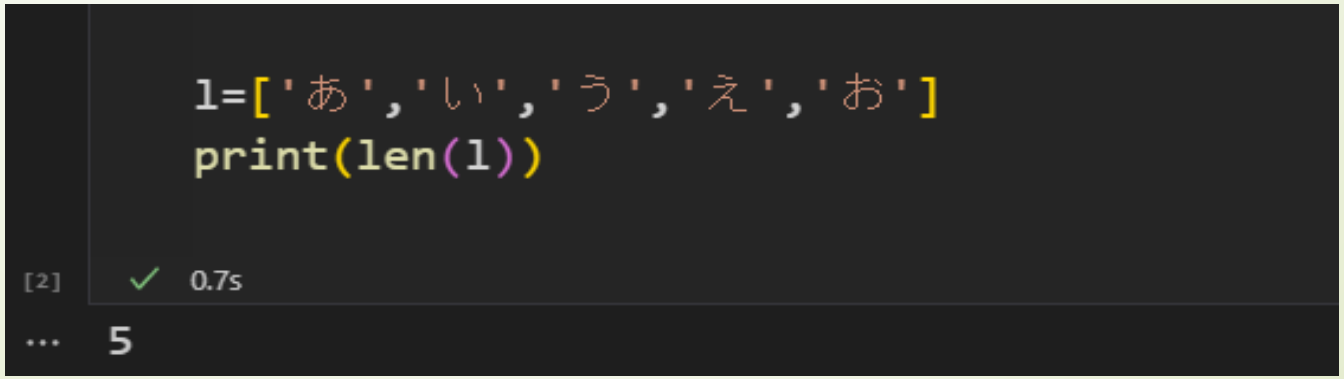
P84~P104　日高明日翔

2.2.4 リスト要素の合計と要素数の取得

リストの要素の合計を取得するにはsum関数を使います。文字列を格納しているリストには使えません。



リストの要素の個数を取得するにはlen関数を使います



2.2.5リスト要素の追加、削除、変更

一度定義したリストに要素を追加したいときはappend関数を使います。

「リスト名.append(追加したい要素)」でリストに要素を追加できます。

```
l=[1,2,3,4,5,6,7,8,9]
l.append(10)
print(l)
```

[5] ✓ 0.6s

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

リストから要素を削除したいときはremove関数を使います。

「リスト名.remove(削除したい要素)」でリストの要素を削除できます。

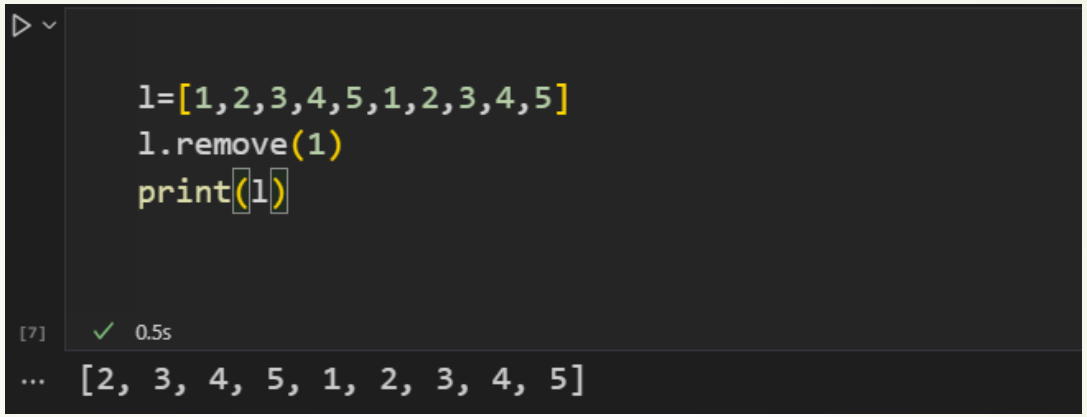
```
l=[1,2,3,4,5,6,7,8,9,10]
l.remove(10)
print(l)
```

[6] ✓ 0.3s

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

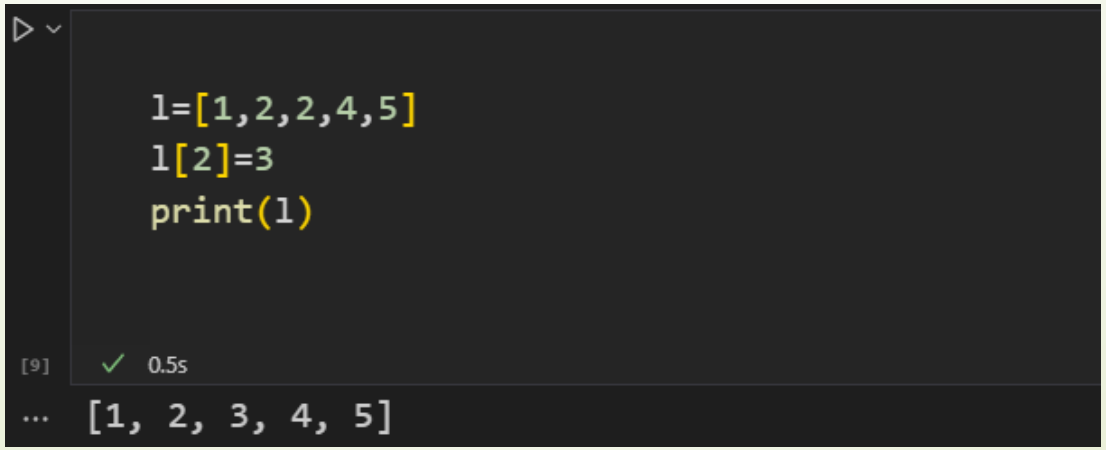
2.2.5リスト要素の追加、削除、変更

削除したい値 x と等しい値がリスト内に複数あった場合、最初の要素を消去します。



コード内の特定の要素を変更したい場合添え字を指定して代入します。その時は

「リスト[変更要素の添え字]＝変更後の値」と書きます



2.2.6 高度な要素の指定

スライスという構文を使用すると連続した範囲にある要素を参照できます。

A:　添え字がA以上の要素を返す

:B　添え字がB未満の要素を返す

```
l=[1,2,3,4,5]
print(l[1:])   #添え字が1以上の要素を参照
print(l[:3])   #添え字が3未満の要素を参照
print(l[1:3])  #添え字が1以上3未満の要素を参照
```

[10] ✓ 0.5s

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

スライスは文字列等にも使えます。

```
s='python'
print(s[1:4])
```

[11] ✓ 0.4s

```
yth
```

2.3.1 ディクショナリの特徴

リストは添え字で要素を管理しているため各要素のデータが持つ具体的な意味合いが分かりにくくなってしまう、というデメリットがあります。

Ex)期末試験の点数をまとめたいがリストだとなんの点数かわからない

このような悩みを解決するにはディクショナリというコレクションが有効です。

ディレクションはそれぞれの要素をキーで管理する。

ディクショナリの定義は「変数={キー1:要素,キー2：要素,・・・}」

```
score={'math':60,'English':70,'Science':65}
print(score)
```

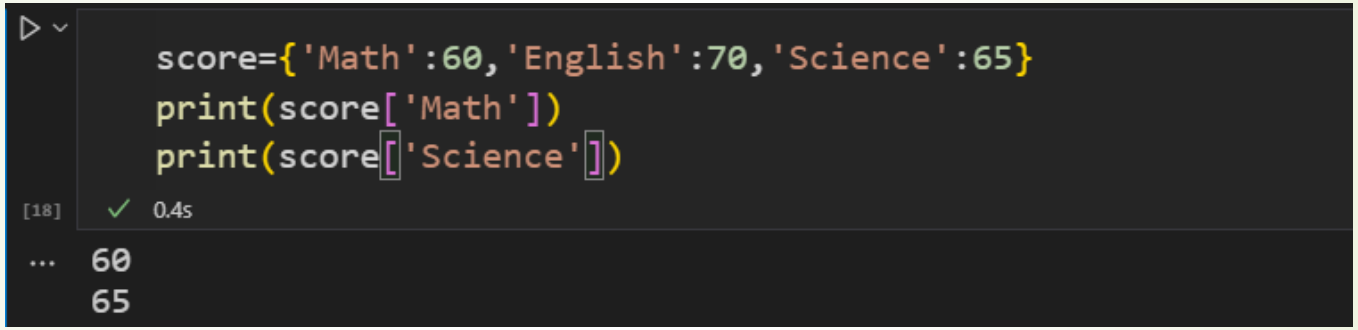
[13] ✓ 0.4s

```
{'math': 60, 'English': 70, 'Science': 65}
```

2.3.3 ディクショナリの要素の参照

ディクショナリの要素を指定するには要素に設定したキーを使います。

「 ディクショナリ名[キー] 」で参照できます。



2.3.4 ディクショナリ要素の追加と変更

ディクショナリの要素を追加、変更したい場合は次のように書きます。

追加する場合：「ディクショナリ名[新しいキー]＝新しい値」

変更する場合：「ディクショナリ名[変更する要素のキー]＝変更後の値」

全く同じ構文なのでキーをしっかり把握する必要がある。

```
score={'Math':60,'English':70,'Science':65}
score['Japanese']=90    #要素の追加
score['English']=50     #要素の変更
print(score)
```

[19] ✓ 0.4s

```
{'Math': 60, 'English': 50, 'Science': 65, 'Japanese': 90}
```

2.3.5 ディクショナリの要素の削除
2.3.6リストとディクショナリの比較

ディクショナリの要素を削除するにはdel文という特殊な構文を用います。

「 del ディクショナリ名[削除したい要素のキー]」で削除できます。

```
score={'Math': 60, 'English': 50, 'Science': 65, 'Japanese': 90}
del score['Japanese']
print(score)
```

[20] ✓ 0.8s

```
{'Math': 60, 'English': 50, 'Science': 65}
```

＜リストとディクショナリの違い＞

リスト：単にデータをまとめて管理したい場合や順序を持つ複数のデータに最適

ディクショナリ：順序を持たない複数のデータに見出しを付けたい場合に最適

ディクショナリコラム

<ディクショナリの順序>

先でディクショナリは順序を持たないといったがPython 3.7以降では追加した順番に要素を取り出せるようになった。ただPython 3.6以前では使えないため、基本的には順序を持たないと覚える。

<ディクショナリの合計>

ディクショナリの要素の合計はリストと違い直接sum関数は使えません。

「ディクショナリ名.values()」でキーを除いた、値だけからなる集まりを取得することが出来ます。

```
score={'Math': 60, 'English': 50, 'Science': 65, 'Japanese': 90}
print(sum(score.values()))  #score.valuesで(60,50,65,90)を取得できる
```

[21] ✓ 0.5s

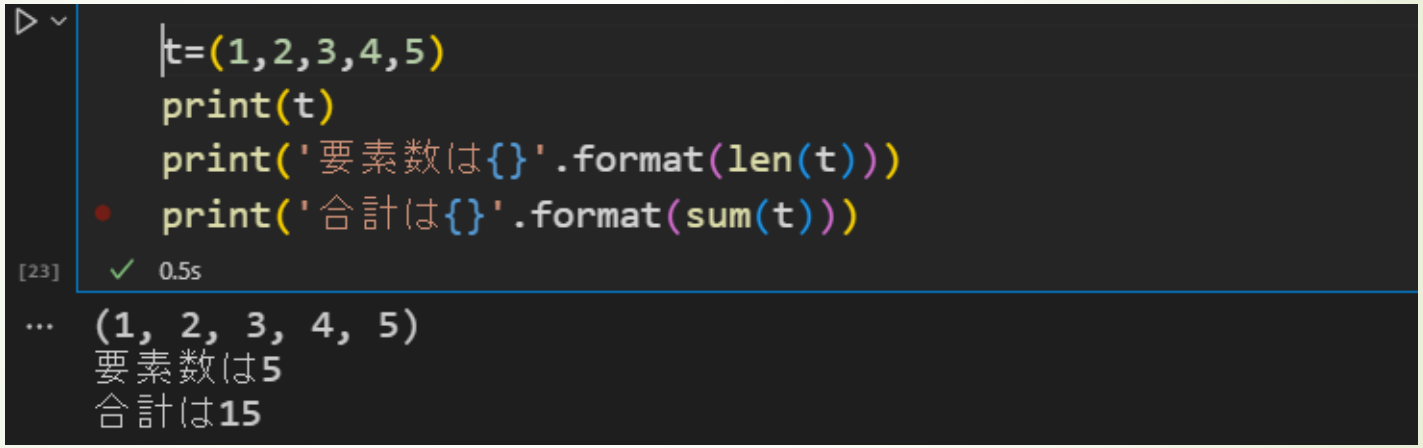
```
265
```

2.4.1 タプル

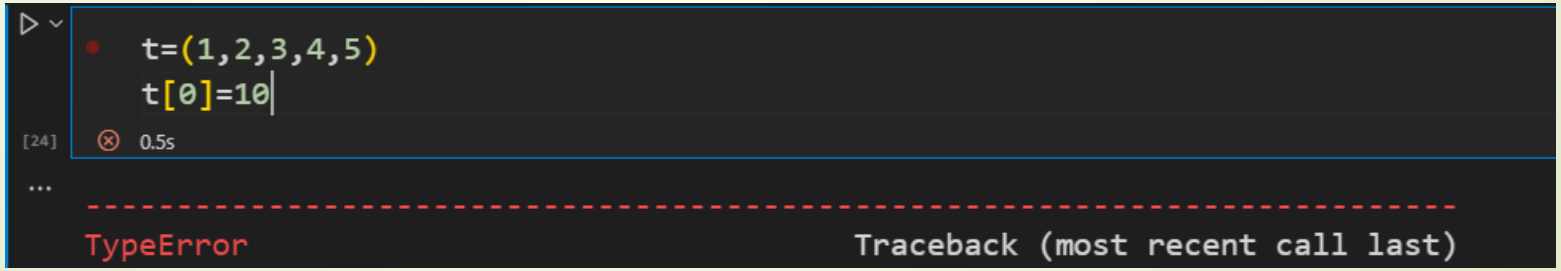
タプルとはリストとほぼ同じコレクションです。唯一、「要素の追加、変更、削除できない。」という点が違います。タプルの定義は

「 変数＝(値1,値2,値3,・・・) 」のようにします。

タプルはリストとほぼ同じなので、len関数、sum関数が使えます。



要素を変えようとするとエラーが出ます



2.4.1 タプル

タプルの要素を変えようとするとエラーが出ます。

```
t=(1,2,3,4,5)
t[0]=10
```

[24] 0.5s

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_3012/2719496467.py in <module>
      1 t=(1,2,3,4,5)
----> 2 t[0]=10

TypeError: 'tuple' object does not support item assignment
```

タプルは意図としない書き換えのリスクを抑えることが出来ます。

非常に似ているリストとタプルをシーケンスと総称されることがあります。

2.4.1　タプル

タプルの要素数が1の場合次のようなことが起こる場合があります。

```
t=('Math')
print(type(t))  #tの型はタプルのはず
```

```
[26]  ✓ 0.8s
...  <class 'str'>
```

それを防ぐためには値の後ろにカンマをつけます。

```
t=('Math',)
print(type(t))  #tの型はタプルのはず
```

```
[27]  ✓ 0.8s
...  <class 'tuple'>
```