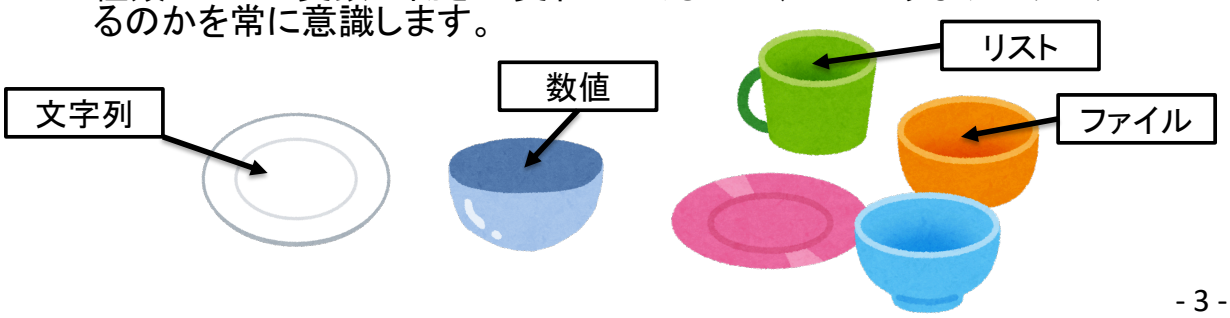
第2回
変数とデータ型

目次

変数

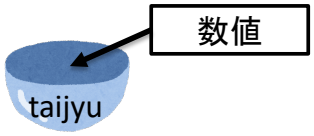
- 変数とは、プログラムで使用するデータを一時的に保存するための入れ物です。
  - 料理に例えれば、野菜などの材料を一時的に入れておくボウルや小皿のようなものです。
  - 複雑なプログラムでは、たくさんのデータを使うため、たくさんの変数を利用します。

- 変数には種類がある
  - 料理で使う入れ物にも、ボウルや小皿、大皿などの種類があるように、変数にも入れるデータによって種類があります。
  - 種類によって変数の働きが変わってくるので、どのようなデータが入っているのかを常に意識します。

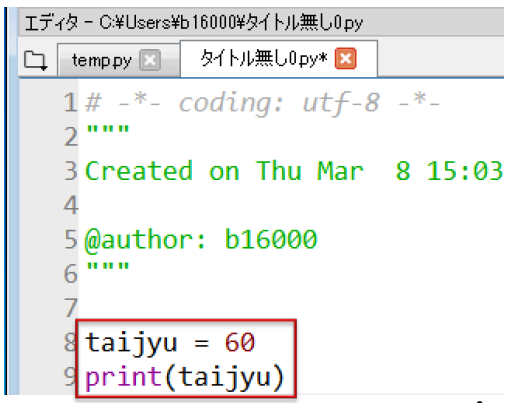


変数を作る(1)

- 変数名
  - 変数には名前を付けます。変数に付けた名前のことを変数名と呼びます。



- 変数の作成と代入
  - pythonでは、「変数名 = データ」と記述すると、指定した変数名を持つ変数が作成され、データを変数に入れることができます。データを変数に入れることを「代入」と呼びます。すでに作成済みの変数の場合は、代入のみが行われます。
  - 「=」を代入演算子と呼びます。
- 例題
  - Spyderのメニューから[ファイル]-[新規ファイル]を選択して、あたらしいソースファイルを作成します。
  - 右の8、9行目を入力し、保存します。その際、ファイル名は変更してもしなくても自由で構いません。

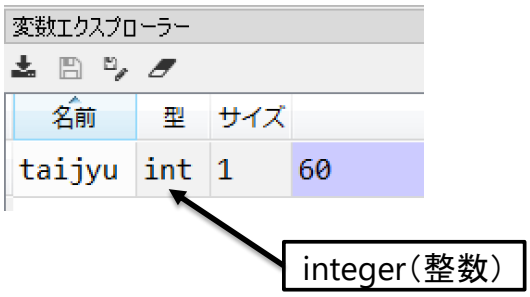


変数を作る(2)

- 実行する
  - プログラムを実行すると、コンソールに「60」と表示されます。print関数の引数として変数taijyuを指定したので、taijyuに入っている値が表示されました。

```
In [1]: runfile('C:/Users/b16000/タイトル無し0.py',
wdir='C:/Users/b16000')
60
```

  - Spyderの右上の情報ウインドウの「変数エクスプローラー」タブをクリックして開くと、実行したプログラムで作成した変数の名前と入っている値が確認できます。



変数を使って計算する

- 変数に入っている値を使って計算を実行することができます。
  - 例:taijyu変数に入っている値に5を足した値を表示する。
    - この例では、taijyu変数に入っている値に変化はありません。

```
8 taijyu = 60
9 print(taijyu + 5)
```

  - 例:taijyu変数に入っている値に5を足した値を、新たなtaijyu変数の値として代入する
    - この例では、taijyu変数に入っている値が5増えます。

```
 8 taijyu = 60
 9 taijyu = taijyu + 5
10 print(taijyu)
```

課題2-1：BMIの計算

- BMIを計算する
  - BMIとは、Body Math Indexの略で、体重と身長から太っているか痩せているかを判断する指標です。
    - BMIが25.0以上：太っている
    - BMIが18.5以上25.0未満：普通
    - BMIが18.5未満：痩せている
  - BMIの計算式は　BMI = 体重（kg）÷身長（m）÷身長（m） です。
    - 下記の囲みを追加して、さらに[ここを考える]を完成させて、BMIを計算してください。

```
 8 taijyu = 60
 9 print(taijyu + 5)
10 shincho = 1.6
11 bmi = [ここを考える]
12 print(bmi)
```

    - 身長や体重を自分や架空の数値に置き換えて、計算してみてください。

変数に文字列を代入する

- 変数には、数値だけでなく文字列も代入できます。
  - 文字列は必ず　"文字列" や、 '文字列' のように、引用符で囲みます。
    - 'の引用符のことをシングルクオーテーション、"の引用符のことをダブルクオーテーションと呼びます。
    - 左と右の引用符の種類は必ず一致させる必要があります。

```
 8 taijyu = 60
 9 print(taijyu + 5)
10 shincho = 1.6
11 bmi = taijyu / shincho / shincho
12 print(bmi)
13 name = "太郎"
14 print(name)
```

  - 数値が入っている変数も、文字列が入っている変数も、変数名だけでは判断できないので、何を入れたかよく覚えておく必要があります。

組み込みのデータ型

- Pythonでは、数値や文字列以外にも、様々なデータ型（データの種類）があらかじめ用意されています。これらを組み込みのデータ型と呼びます。

データ型	説明
数値（int, float）	数値は、さらに厳密には整数型（int）、浮動小数点型（実数型：float）に内部で別れています。
文字列（str）	文字を順番に並べたもので、テキスト型とも呼ばれます。
真偽（bool）	True（真）とFalse（偽）という2値のみを持つデータ型
リスト（list）	複数のデータを順番に並べて利用するためのデータ型です。設定したデータを並べ替えたり、値を変更したりできます。
タプル（tuple）	リストと同様に、複数のデータを順番に並べて利用するためのデータ型ですが、一度設定したら変更ができないのがタプルです。
ディクショナリ（dict）	辞書型とも呼ばれ、リストと似ていますが、順番に並べるのではなく、キーと呼ばれる索引を使って管理します。
セット（set）	リストに似ていますが、同じデータの重複を許しません。

数値型の詳細

- 数値型は演算子を使った四則演算が可能です。
  - 加算＋　減算－　乗算＊　除算／

```
15 a = 10 % 4
16 print(a)
17 b = 4 ** 2
18 print(b)
```

- その他にも演算子が用意されています。
  - 割り算の余りを求める%　累乗＊＊

- 演算子の優先順位
  - 演算子には加算・減算よりも乗算・除算を優先するという優先順位があるので、このルールに反して優先したい計算がある場合は()で囲む必要があります。

1+2を先に計算したい場合は？

```
19 c = 1 + 2 * 4
20 print(c)
```

- 複合演算子
  - 四則演算と代入を併せ持つ演算子として複合演算子があります。
    - 加算して代入＋＝　減算して代入－＝
      乗算して代入＊＝　除算して代入／＝

```
21 d = 0
22 d = d + 2
23 print(d)
24 d += 2
25 print(d)
```

文字列型の詳細

- 文字列は、複数の文字が連続したデータ型です。
  - 左端の文字から0番目、1番目・・・と番号が割り振られており、これをインデックスと呼びます。

インデックス	0	1	2	3	4	5
文字	P	y	t	h	o	n

  - 文字列変数の末尾に角括弧[]を付けて、その中にインデックスを指定して、特定の文字を取り出すことができます。

```
26 name = "太郎"
27 print(name[1])
```

何が表示される？

  - 文字列の一部に"や'を含めるには？

```
27 sample = "I am 'Taro'"
28 print(sample)
```

'を含めたいならば全体を"で囲む

  - 複数行からなる文字列を代入するには？

```
29 shizuoka = """静
30 岡
31 県"""
32 print(shizuoka)
```

全体を"""
"""で囲む

文字列に対する演算子

- 文字列同士の＋は連結を表す
  - 数値では＋や＊は加算や乗算でしたが、文字列同士の＋は複数の文字列を繋げる働きをします。

```
34 txt = "1234"
35 txt = txt + "5678"
36 print(txt)
```

"1234"と"5678"は数値ではなくて文字列なので？

  - 文字列と数値を＋しようとするとエラーになります。

```
34 txt = "1234"
35 txt = txt + 5678
36 print(txt)
```

  - 文字列＊数値は文字列が数値の数だけ繰り返します。

```
37 txt = "はい" * 10
38 print(txt)
```

文字列と数値の相互変換

- "1234"のような数字からなる文字列を1234という数値に、またその逆を行う関数が用意されています。
  - 下記は数値と文字列を＋しようとしており、エラーになります。

```
39 txt = "1234"
40 num = 1 + txt
41 print(num)
```

  - int関数：文字列を整数に変換する

```
39 txt = "1234"
40 num = 1 + int(txt)
41 print(num)
```

  - float関数：文字列を実数に変換する

```
39 txt = "1234"
40 num = 1 + float(txt)
41 print(num)
```

  - str関数：数値を文字列に変換する

```
39 txt = "1234"
40 num = str(1) + txt
41 print(num)
```


文字列の長さと包含チェック

- 文字列の長さを求めるlen関数

```
42 address="静岡県焼津市"
43 nagasa = len(address)
44 print(nagasa)
```

- in演算子
  - in演算子を使うことで、ある文字列に特定の部分文字列が含まれているかを調べることができます。結果は含まれていればTrue、含まれていなければFalseという真偽型で返却されます。

  構文：　部分文字列　in　全体文字列

```
42 address="静岡県焼津市"
43 kekka = "焼津" in address
44 print(kekka)
```

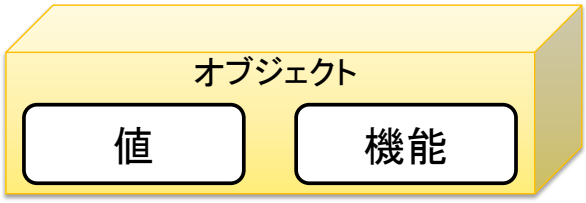
True

```
42 address="静岡県焼津市"
43 kekka = "浜松" in address
44 print(kekka)
```

False

データ型とオブジェクト

- オブジェクトとは？
  - なんらかの値と、自分自身に対する特別な機能を合わせ持ったプログラムを構成する部品のことです。
  - オブジェクトは日本語に訳すと「モノ」になります。現実世界の様々なモノは、何らかの値（エアコンであれば設定温度など）と、機能（エアコンであれば設定温度を上げる・下げるなど）を持っています。値と機能をひとまとめにしたオブジェクトという部品を組み合わせたプログラム開発手法を、オブジェクト指向プログラミングと呼びます。



- Pythonのデータは全てがオブジェクト
  - 実は、Pythonで用意されている全てのデータはオブジェクトなのです。つまり、値だけでなく自分自身に対する機能も合わせ持っています。
  - Pythonでは、オブジェクト内で値を保持するための変数をインスタンス変数、機能のことをメソッドと呼びます。
  - 関数とメソッドは似ていますが、メソッドはある特定のオブジェクト専用の関数という位置づけになります。

文字列型のメソッド（1）

- データに対してメソッドを呼び出すには？
  - データの末尾にドットで区切ってメソッド名を指定します。データは、変数に代入された状態でも、代入前でもどちらでも大丈夫です。
    構文：データ.メソッド名(引数)

- 文字列型のメソッドの例：
  - 文字列を大文字に変換するupperメソッド

```
45 txt = "shizuoka".upper()
46 print(txt)
```

```
45 txt = "shizuoka"
46 txt = txt.upper()
47 print(txt)
```

  - 文字列を小文字に変換するlowerメソッド

```
44 txt = "SHIZUOKA".lower()
45 print(txt)
```

```
45 txt = "SHIZUOKA"
46 txt = txt.lower()
47 print(txt)
```

文字列型のメソッド(2)

- 文字列の検索ができるindexメソッド
  - ある文字列に特定の部分文字列が含まれているかを調べて、その開始インデックスを検索することができます。

```
47 address="静岡県焼津市"
48 print(address.index("県"))
```

インデックスは0から始まることに注意

  - 見つからないとエラーになるため、あらかじめin演算子で含まれているかチェックしてから使用します。

- 文字のカウントができるcountメソッド

```
49 txt = "静岡県静岡市"
50 print(txt.count("静"))
```

いくつが出力される？

- 文字列の置き換えができるreplaceメソッド
  - 構文：データ.replace("検索文字列", "置換文字列")

```
49 txt = "静岡県静岡市"
50 print(txt.replace("静岡市", "三島市"))
```


リスト

- リストは複数のデータを順番に入れておくことのできるデータ型です。
  - 文字列型では文字を順番に入れておくことができましたが、リストでは、どのようなデータ形式でも要素として持つことができます。
  - 文字列型と同様に、要素の順番はインデックスで指定ができます。

インデックス	0	1	2	3	4	5
要素	"太郎"	60	165	"男性"	1996	7

  - 文字列型と同様に＋や＊、inなどの演算子やメソッドが使えます。

- リスト用に、新しくソースファイルを作成する。
  - Spyderのメニューから[ファイル]-[新規ファイル]を選択して、あたらしいソースファイルを作成します。

リストを定義する

- リストを定義するには角括弧[]を使う
  - 構文：変数名 = [要素, 要素, 要素, ・・・]
  - インデックスを指定してリストの要素を取り出す際にも、文字列と同じように[]を使う
- 例：

インデックス	0	1	2	3	4	5
要素	"太郎"	60	165	"男性"	1996	7

```
 8 a = ["太郎", 60, 165, "男性", 1996, 7]
 9 print(a)
10 print(a[3])
```

何が出力される？

  - インデックスとしてマイナスの整数を指定すると、最後から数えた要素を取り出せます。

```
10 print(a[-1])
```

何が出力される？

スライス

- スライスという機能を使うと、リストの複数の要素をインデックスの範囲を指定して取り出すことができます。
  - スライスの範囲指定は、開始インデックスと終了インデックスに1を加えた整数をコロン(:)で区切って指定します。
    構文：変数名[開始インデックス:終了インデックス＋1]
  - 例：リストの一部を取り出して、別のリストを作る例

インデックス	0	1	2	3
要素	"太郎"	60	165	"男性"

ここだけ取り出して別のリストbを作る

```
 8 a = ["太郎", 60, 165, "男性", 1996, 7]
 9 print(a)
10 print(a[-1])
11 b = a[1:3]
12 print(b)
```

  - 開始インデックスが先頭の場合、開始インデックスの指定を省略できます。同様に、終了インデックスが末尾の場合も省略できます。

```
11 b = a[:3]
12 print(b)
```

```
13 b = a[3:]
14 print(b)
```

何が出力される？

リストへの要素の追加

- 定義済みリストに要素を追加する様々な方法が用意されています。
  - ＋演算子を使った方法
    - 文字列の連結と同様に、リストにおける＋演算子は連結を表します。
    - 連結できる対象はリスト同士です。下記の左の例では、エラーになります。

×
```
16 a = [1, 2, 3, 4]
17 b = a + 1
18 print(b)
```

○
```
16 a = [1, 2, 3, 4]
17 b = a + [1]
18 print(b)
```
↓
```
[1, 2, 3, 4, 1]
```

○
```
16 a = [1, 2, 3, 4]
17 b = a + [1,-1]
18 print(b)
```
↓
```
[1, 2, 3, 4, 1, -1]
```

  - appendメソッドを使うと、自分自身に要素を追加できます。
    - 構文：リスト.append(追加要素)
    - 追加要素として、リストを指定した場合には、リスト全体が要素として追加されるので注意が必要です。リストを要素としてではなく、中身を追加したい場合は、extendメソッドを使います。

```
16 a = [1, 2, 3, 4]
17 a.append(5)
18 print(a)
```
↓
```
[1, 2, 3, 4, 5]
```

```
16 a = [1, 2, 3, 4]
17 a.append([5, 6])
18 print(a)
```
↓
```
[1, 2, 3, 4, [5, 6]]
```

extendを使うと？

リストの要素の上書きと削除

- 要素の上書き
  - 定義済みのリストの要素を上書きするには、インデックスを指定して代入します。
  - 例：
```
16 a = [1, 2, 3, 4]
17 a[1] = 100
18 print(a)
```
⇒ `[1, 100, 3, 4]`

- 要素の削除
  - リストの要素を削除するには、del文を使います。
    構文：del リスト[インデックス]
  - 例：
```
16 a = [1, 2, 3, 4]
17 del a[1]
18 print(a)
```
⇒ `[1, 3, 4]`

リストの要素の検索その他

- 文字列と同様に、要素の包含チェックとしてin演算子、検索にindexメソッドが使えます。
  - 例:
```
16 a = [300, 100, 400, 200]
17 b = 300 in a
18 print(b)
19 c = a.index(300)
20 print(c)
```

何が出力される？

- max関数とmin関数を使うと、要素の最大値、最小値を求めることができます。
  - 例:
```
16 a = [300, 100, 400, 200]
17 b = min(a)
18 print(b)
```

```
16 a = [300, 100, 400, 200]
17 b = max(a)
18 print(b)
```

- sort関数を使うと要素を小さい順に並べ替えできます。reverse関数を使うと要素の並びを逆にできます。
  - 例:
```
16 a = [300, 100, 400, 200]
17 a.sort()
18 print(a)
```

```
16 a = [300, 100, 400, 200]
17 a.reverse()
18 print(a)
```


ディクショナリ（辞書）

- ディクショナリは複数のデータをキーと呼ばれる見出しを使って管理できるデータ型です。
  - インデックスで要素を指定するのではなく、キーと呼ばれる数値や文字列で要素を指定できるのが特徴です。

キー	値（バリュー）
住所	静岡県静岡市
氏名	山田太郎
年齢	26

- ディクショナリ用に、新しくソースファイルを作成する。
  - Spyderのメニューから[ファイル]-[新規ファイル]を選択して、あたらしいソースファイルを作成します。

ディクショナリを定義する

- ディクショナリを定義するには波括弧{}を使う
  - 構文：変数名 = {キー1: 値1, キー2: 値2, ・・・}
  - キーと値の組みはコロン(:)で区切って記述します。
  - 値を取り出す際は[]の中に取り出したい値のキーを指定します。
  - 例：

キー	値(バリュー)
住所	静岡県静岡市
氏名	山田太郎
年齢	26

```
8 a = {"住所":"静岡県静岡市", "氏名":"山田太郎", "年齢":26}
9 print(a["氏名"])
```

何が出力される？

  - 存在しないキーを指定すると、エラーになります。

キーと値の追加、削除

- キーと値の追加
  - ディクショナリに新たなキーと値を追加するには、新しいキーを添えて代入を行います。
  - 例：

```
 8 a = {"住所":"静岡県静岡市", "氏名":"山田太郎", "年齢":26}
 9 a["体重"] = 60
10 print(a)
```

- キーを使って値を削除する
  - 文字列やリストと同様に、del文を使います。
  - 例：

```
 8 a = {"住所":"静岡県静岡市", "氏名":"山田太郎", "年齢":26}
 9 del a["年齢"]
10 print(a)
```

キーの存在チェックと一覧

- 特定のキーが存在しているか確認するにはin演算子を使います。
  - 例：
```
 8 a = {"住所":"静岡県静岡市", "氏名":"山田太郎", "年齢":26}
 9 b = "年齢" in a
10 print(b)
```


True

- ディクショナリ内のキーの一覧を取得するにはkeysメソッドを使います。
  - ただし、取得した一覧はイテレータという特殊な形式なので、例えばリスト形式で取得したければ、list関数でリストに変換します。
  - 例：
```
 8 a = {"住所":"静岡県静岡市", "氏名":"山田太郎", "年齢":26}
 9 b = a.keys()
10 c = list(b)
11 print(c)
```

タプル

- タプルはリストに似ていますが、一度定義するとその後は要素を追加したり変更することができません。
- タプルの定義
  - タプルを定義するためには丸括弧()で要素を囲みます。
    例：
```
13 a = (1, 2, 3, 4, 5)
14 print(a)
15 print(a[2])
```

- タプルの利点
  - タプルはリストに比較して自由度は低いですが、あらかじめサイズが決まっている分、メモリ効率がよくなります。
  - リストと違って、ディクショナリのキーとして使用することも可能です。

課題2-2：まとめ問題

```
 8 kanji = ["静岡", "浜松", "沼津"]
 9 yomi = {"静岡":"しずおか", "浜松":"はままつ", "沼津":"ぬまづ"}
10 n = input("0～2までの数字を入力してEnterキーを押してください");
```

- 上記コードが与えられたとします。nに0が入力されたならば「しずおか」、1ならば「はままつ」、2ならば「ぬまづ」と出力してください。
  - 解説とヒント：
    - input関数は、キーボードから文字を入力できる関数です。上記の例では、入力された文字がnに代入されます。
    - nに代入されるのは数値ではなく文字なので、リストのインデックスとして用いるためには数値に変換する必要があります（int関数を使う）。