

プログラミング初級 (Python)

データ型と変数

早稲田大学グローバルエデュケーションセンター

Hello, world!

合抱の木も毫末より生じ、九層の台も累土より起こり、
千里の行も足下より始まる。

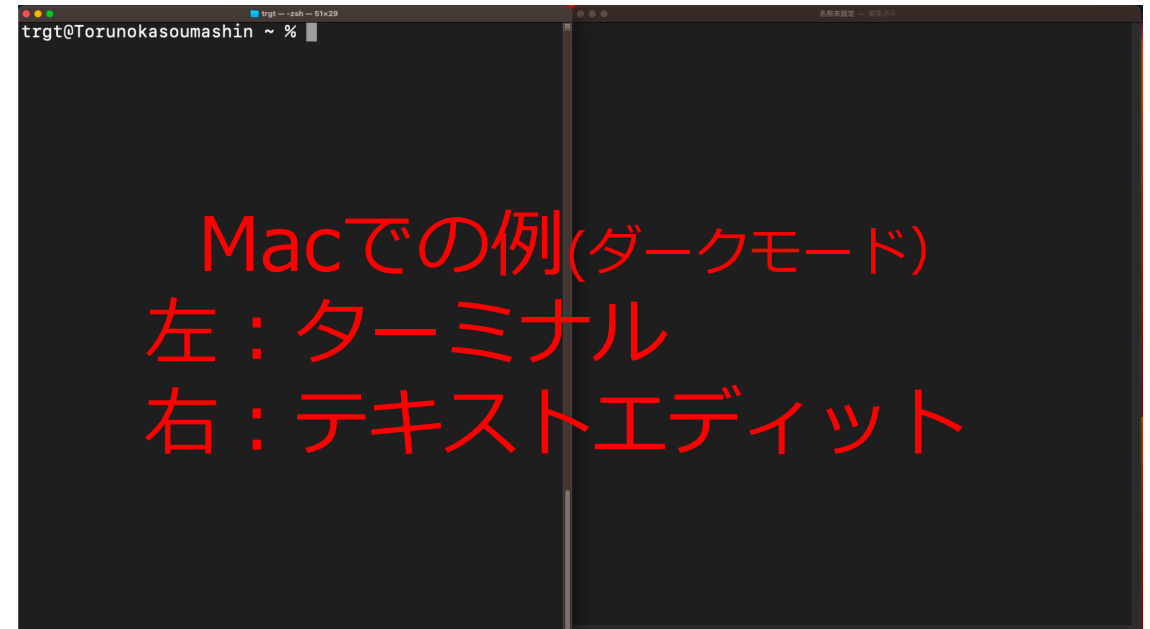
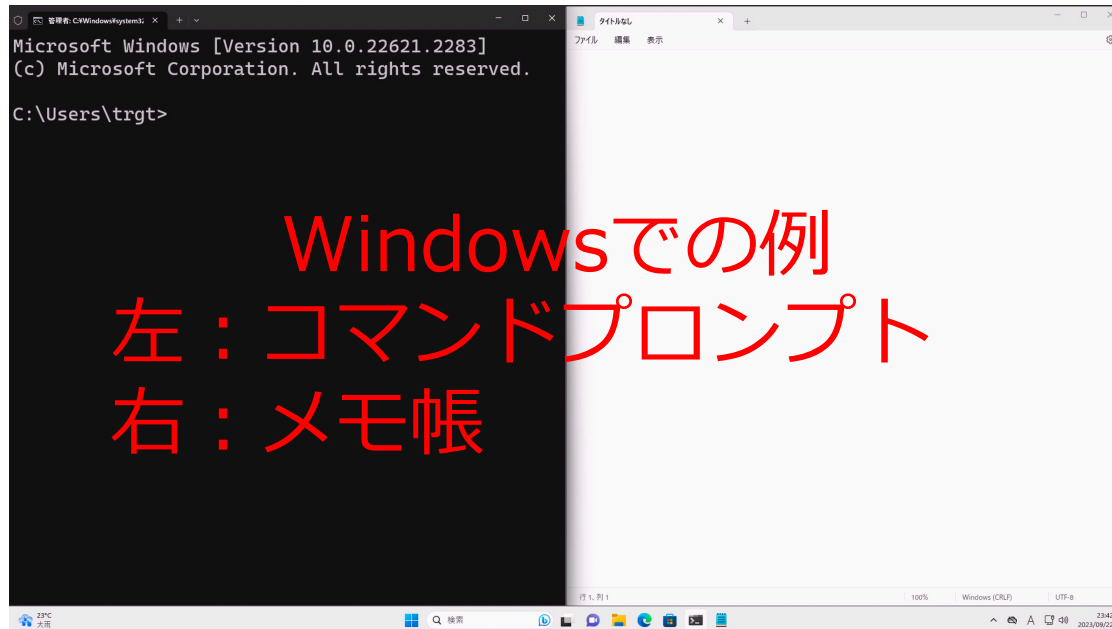
— 老子

Hello, world! –初めの第一歩、千里の道も一歩から– 共通

1. ターミナル系ソフトを起動。
2. 任意のテキストエディタを起動。
3. 両者の配置やサイズを調整。
例 左にターミナル系ソフト
右にテキストエディタ
4. 文字サイズを調整。

Windowsでは、並べたい2つのソフトを起動した状態で、例えば左に配置したいソフトを選択し、アクティブにした状態で **[Windowsキー] + [←]** キーを入力すると、そのソフトが画面の左半分に配置される。自動的に残りのソフトが選択できる状態になるので実行するとそのソフトが右半分に配置される。

Macでは、並べたい2つのソフトを起動した状態で、例えば左に配置したいソフトのウィンドウ上部にある3つ目のボタンを長押しすると「ウィンドウを画面左側にタイル表示」とあるのでそれをドラッグしながら選択し、自動的に残りのソフトが選択できる状態になるので実行するとそのソフトが右半分に配置される。



Hello, world! –初めの第一歩、千里の道も一歩から– ■編

5. 四角内のソースコードをテキストエディタで作成。数字は行数を表す。

```
1 print("Hello, world!")
```

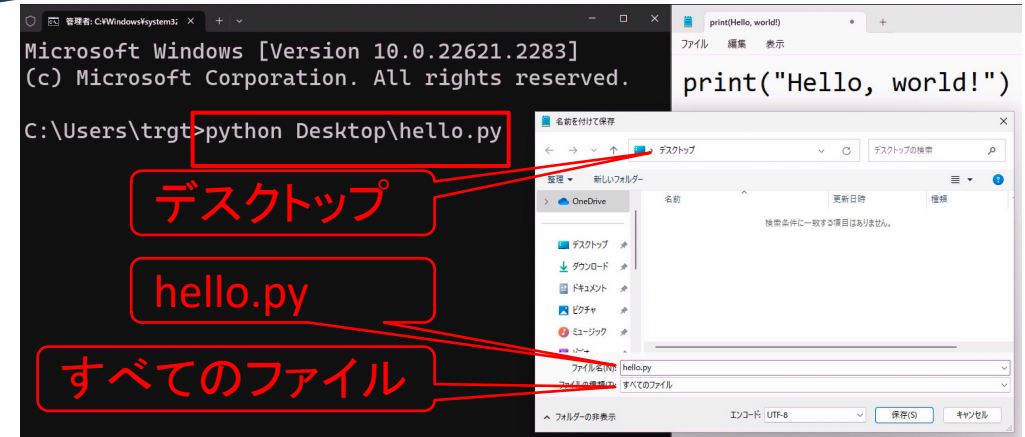
半角スペースあり。全角スペースは無効。エラーとなる。

[ファイル] → [保存]

6. ファイル名を「**hello.py**」とし、テキストエディタを**操作**してプログラムを**デスクトップ**に**保存**。

7. コマンドプロンプトを起動するとホームディレクトリ(trgt)が現在の基点。

8. 下記のコマンドでプログラムを実行する。このコマンドは**デスクトップ**に**保存したPythonプログラム**を**Pythonインタプリタ**で**実行**するという意味。入力後、[Enter]キーを押す。



```
>python Desktop¥hello.py
```

>はプロンプト記号を意味するので**入力不要**。

¥記号は\ (バックスラッシュ) となっても良い。どちらになるかはシステムに依存する。

半角スペースあり。以後言及しないがスペースの有無には常に警戒。

Hello, world! –初めの第一歩、千里の道も一歩から– 編

9. プログラムの**実行結果**を確認する。
下記の破線内及び右の赤線内が実行結果である。

```
Hello, world!
```

```
管理者: C:\Windows\System32
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\trgt>python Desktop\hello.py
Hello, world!

C:\Users\trgt>
```

10. 基点をホームディレクトリからデスクトップに変更すれば、プログラムの実行に必要な記述を簡略化できることから、下記のコマンドを実行する。このコマンドは**基点をホームディレクトリからデスクトップにチェンジ**という意味。入力後、[Enter]キーを押す。

```
>cd Desktop
```

11. デスクトップ起点でプログラムを実行。

```
>python hello.py
```

12. プログラムは同じなので**実行結果は不変**。

```
Hello, world!
```

```
管理者: C:\Windows\System32
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\trgt>python Desktop\hello.py
Hello, world!

C:\Users\trgt>cd Desktop

C:\Users\trgt\Desktop>python hello.py
Hello, world!

C:\Users\trgt\Desktop>
```

Hello, world! –初めの第一歩、千里の道も一歩から– 🍏 編

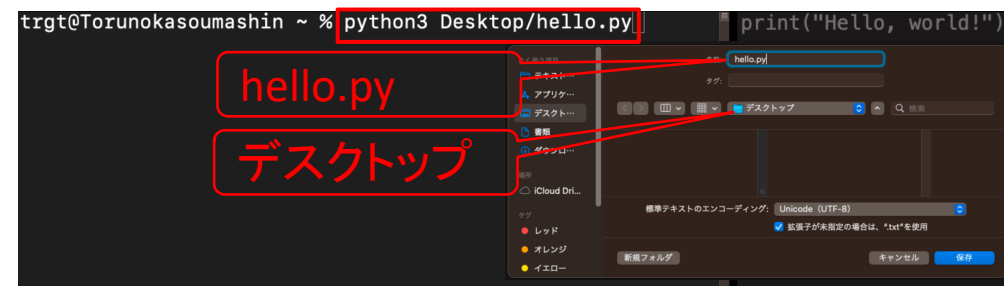
5. 四角内のソースコードをテキストエディタで作成。数字は行数を表す。

```
1 print("Hello, world!")
```

半角スペースあり。全角スペースは無効。エラーとなる。

[ファイル] → [保存]

6. ファイル名を「**hello.py**」とし、テキストエディタを**操作**してプログラムを**デスクトップ**に**保存**。



7. コマンドプロンプトを起動するとホームディレクトリ(trgt, ~)が現在の基点。

8. 下記のコマンドでプログラムを実行する。このコマンドは**デスクトップ**に**保存したPythonプログラム**を**Pythonインタプリタ**で**実行**するという意味。入力後、[Enter]キーを押す。

```
%python3 Desktop/hello.py
```

%はプロンプト記号を意味するので**入力不要**。

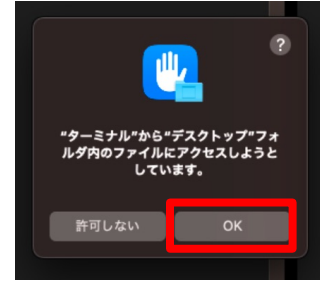
python ではなく、**python3** なことに注意。

半角スペースあり。以後言及しないがスペースの有無には常に警戒。

Hello, world! –初めの第一歩、千里の道も一歩から–  編

9. プログラムの**実行結果**を確認する。アクセス許可が出た場合[OK]を必ず選択。下記の破線内及び右の赤線内が実行結果である。

Hello, world!



```
trgt@Torunokasoumashin ~ % python3 Desktop/hello.py
Hello, world!
trgt@Torunokasoumashin ~ %
```

10. 基点をホームディレクトリからデスクトップに変更すれば、プログラムの実行に必要な記述を簡略化できることから、下記のコマンドを実行する。このコマンドは基点をホームディレクトリからデスクトップにチェンジという意味。入力後、[Enter]キーを押す。

%cd Desktop

11. デスクトップ起点でプログラムを実行。

%python3 hello.py

```
trgt@Torunokasoumashin ~ % python3 Desktop/hello.py
Hello, world!
trgt@Torunokasoumashin ~ % cd Desktop
trgt@Torunokasoumashin Desktop % python3 hello.py
Hello, world!
trgt@Torunokasoumashin Desktop %
```

12. プログラムは同じなので**実行結果は不変**。

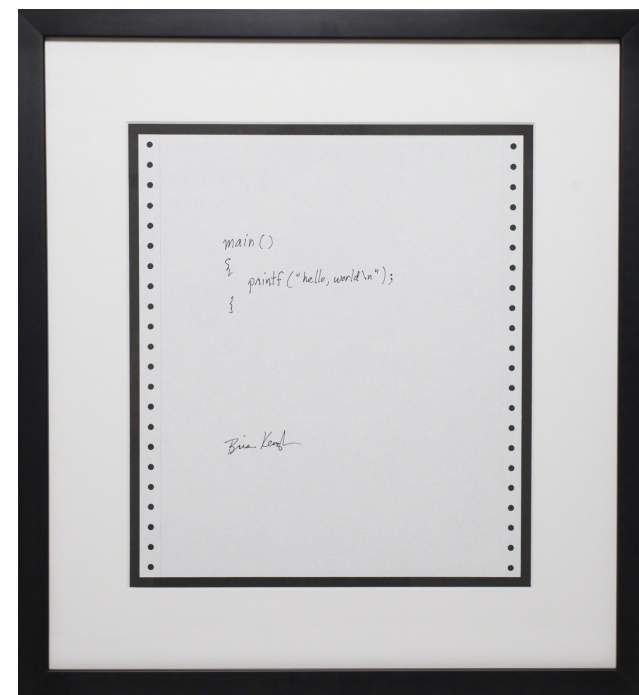
Hello, world!

利用するツールとHello, world!プログラムについて

- **OS標準のターミナル系のソフトとテキストエディタの併用は基本的かつ汎用的なスキル**。業務で使うような組織の管理下にあるPCなどは個人でカスタマイズできない場合あり。応用も効くので習得が望まれる。
- ターミナル系ソフトの機能、テキストエディタの機能、その他の便利機能がまとまっているものが統合開発環境。
- プログラミングにおいて利用する**ツールに制限はなく、プログラミングが行えるのなら何でも良い**。
- Hello, world!と印字するプログラムを初めに作るのはどの言語を用いる場合でも慣例となっている。
- この習慣は、C言語を開発したブライアン・カーニハンとデニス・リッチーの著書「プログラミング言語C(1978年)」から始まったと言われている。

以後、具体的なツールの教示は割愛する。
プログラムのソースと実行結果のみを示す。

ブライアン・カーニハンによる"Hello, world"プログラム (1978)→



冒頭のプログラム(spells.py)の作成と実行

・ソースコード

```
1 spells = [  
2     "Riddikulus!",  
3     "Wingardium Leviosa!",  
4     "Avada Kedavra!",  
5     "Expecto Patronum!",  
6     "Nox!",  
7     "Lumos!",  
8 ]  
9 print(spells[3])
```

・実行結果

Expecto Patronum!

以後、特に言及しないが破線内の表現はテキストエディタ等で作成したプログラムの実行結果を示すこととする。

これは複数候補からハリーポッターの呪文を1つ選んで表示する小さなプログラム。

個々の呪文は、クォートで囲まれた文字の並びになっている。これを**文字列**という。呪文はカンマで区切られ、全体が角括弧([と])で囲まれている。これを**リスト**という。spellsのような単語は、リストに名前を与えて操作できるようにするためのラベルで、**変数**という。

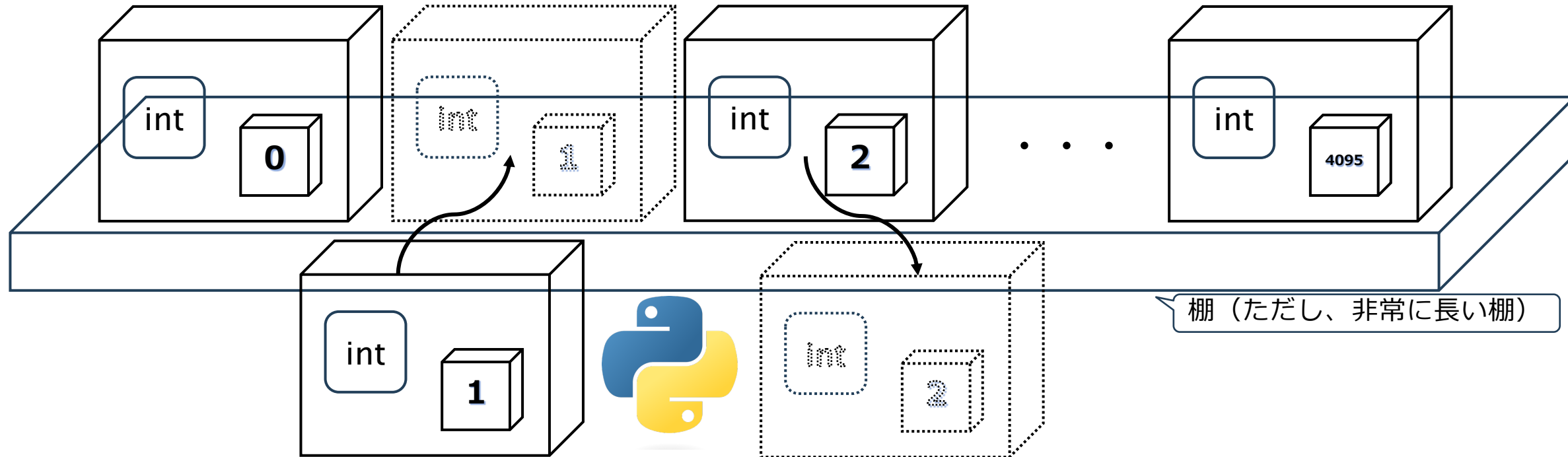
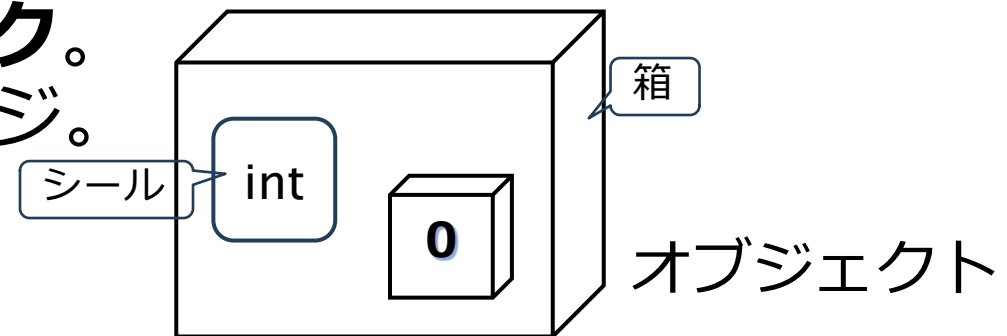
このプログラムの場合、第4の呪文がprint**関数**によって画面に表示される。4番目の呪文を取り出す際にspells[3]と命令するのはリストの先頭を**0番**として数え始めるから。**ではなぜそうするのでしょう？**例えば、呪文の数を知りたい場合、カンマが何個あるかを数えれば話しの早い解決法。その場合、"Riddikulus!"の後のカンマが来て初めて1番とカウントすることになるが、その前の文字がある段階で先頭を1番として数えてしまうとその後は2番と数えて最後に辻褃合わせで1引く処理が必要となる。**このような面倒をなくするため、コンピュータの世界では値の先頭を0番から数える。**

データ型（型）とは？

- コンピュータは、メモリにデータ（命令や値）を読み込んだ書き込んだりしながら、CPUで命令を実行する。
- プログラムはどこ（メモリでの位置）に**何**（データ型や値）があるかを管理する。
- コンピュータの視点では、データ（ここでは**値**）は**ただのビット列**。
- ビット列をそのまま解釈すると値は数でしかないが、他の意味（論理や文字など）も表現したい場合どうすればよいか。
- **値（ビット列）に型を付加**する。
- 型という分類情報があれば値が同じ場合でも別の意味を表現できる。
例 **10進数65**の値は **01000001** のビット列。
 ASCII文字Aの値は **01000001** のビット列。
前者は整数型、後者は文字型と呼ばれる型を有する。このように**型が異なれば同じ値（ビット列）でも別の意味**を表現できる。

Pythonではデータが全てオブジェクト

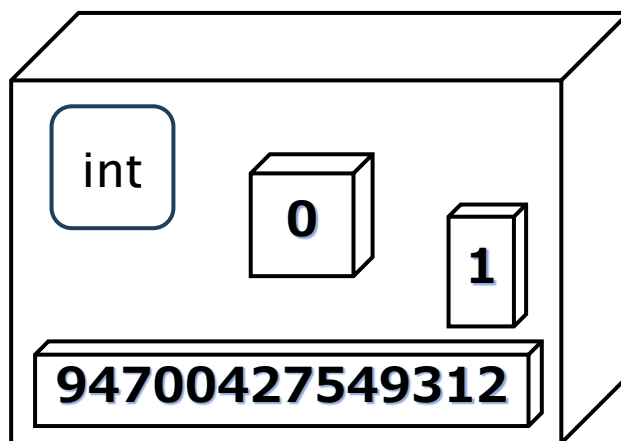
- オブジェクトとは、**型と値(など)のチャンク**。
左図は**整数(int)型と1の値**を有するイメージ。
- メモリを棚に例えると、オブジェクトは箱に例えることができる。
- Pythonはオブジェクトの箱を作り、棚の空いたスペースに置く。
不要になれば、棚からオブジェクトを取り除く。下記は簡略イメージ。


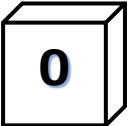
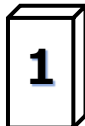
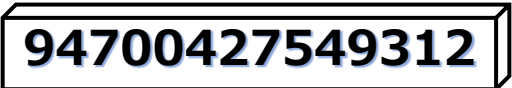


オブジェクトの構成をもう少し詳しく

- Pythonでは、オブジェクトとは、
型、値、参照カウント、ID(など)のチャンク。

オブジェクト



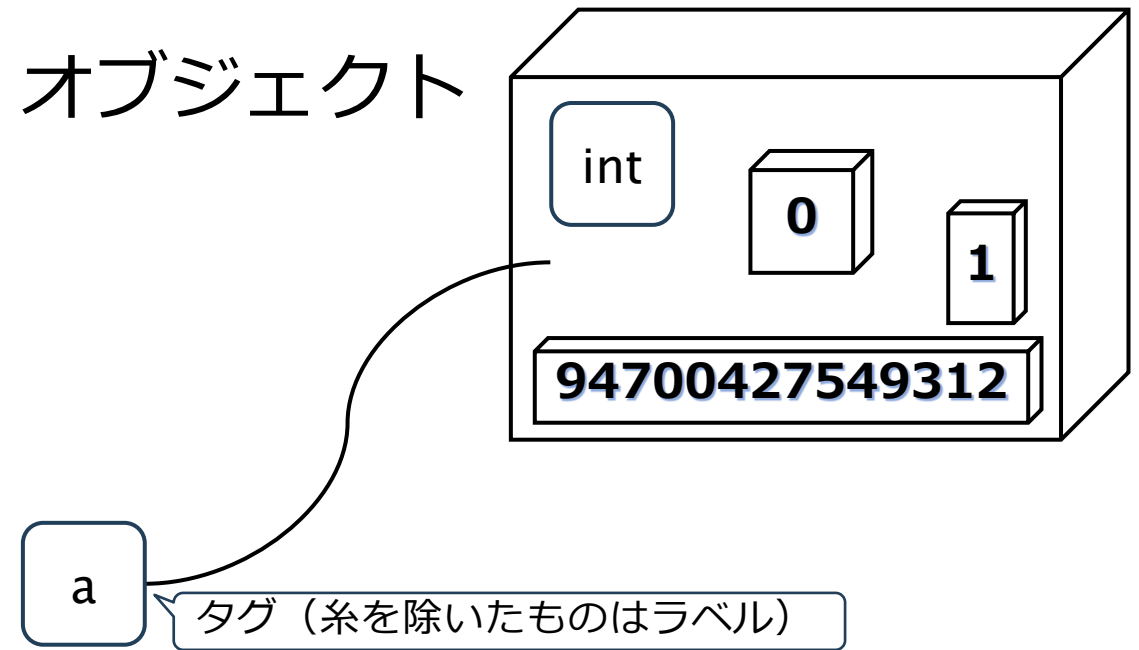
- 値が何を意味するかを定義する**型**。 
- 型に従って解釈できる**値** 
- 当該オブジェクトが何回使われているかを示す**参照カウント** 
- 他のオブジェクトとの区別に使われる**ID** 

Pythonにおける基本のデータ型

呼称	表記	値の例
ブーリアン	<code>bool</code>	<code>True</code> , <code>False</code>
整数	<code>int</code>	<code>12</code> , <code>15000</code> , <code>15_000</code>
浮動小数点数	<code>float</code>	<code>3.14</code> , <code>2.7e5</code>
複素数	<code>complex</code>	<code>3j</code> , <code>5 + 9j</code>
文字列	<code>str</code>	<code>'waseda'</code> , <code>"university"</code> , <code>"""Okuma Shigenobu"""</code>
リスト	<code>list</code>	<code>['Okuma', 'Ono', 'Takada']</code>
タプル	<code>tuple</code>	<code>(2, 4, 6)</code>
バイト	<code>bytes</code>	<code>b'ab\xff'</code>
バイト配列	<code>bytearray</code>	<code>bytearray(...)</code>
集合	<code>set</code>	<code>{3, 5, 7}</code>
frozenset	<code>frozenset</code>	<code>Frozenset(['Elsa', 'otto'])</code>
辞書	<code>dict</code>	<code>{'python': 'pass', 'java': 'failure', 'c': 'pass'}</code>

変数とは？

- ほとんどのプログラミング言語と同様にPythonでも変数を定義できる。
- Pythonにおける**変数とは**プログラムで使うメモリ上の**オブジェクトにつけた名前**。箱というよりかは**ラベル**。
- すなわち、変数はオブジェクトの在処を示す**参照情報**。
- 代入は**オブジェクトと変数の関連づけを行う操作**。イメージは**タグづけ**。
- CやJavaなどの静的型付け言語では変数自体が型を有し、メモリの位置が固定。その位置の値は書き換えられるが、新しい値は同じ型でなければならない。例 `int height=65; char c='A';`
- Pythonでは変数宣言時に型を指定しなくて良い。変数名(ラベル)はどんなオブジェクトでも指すことができ**タグをたどれば型と値が分かる**。例 `height=65; c='A';` このような言語は**動的型付け言語**と呼ばれる。



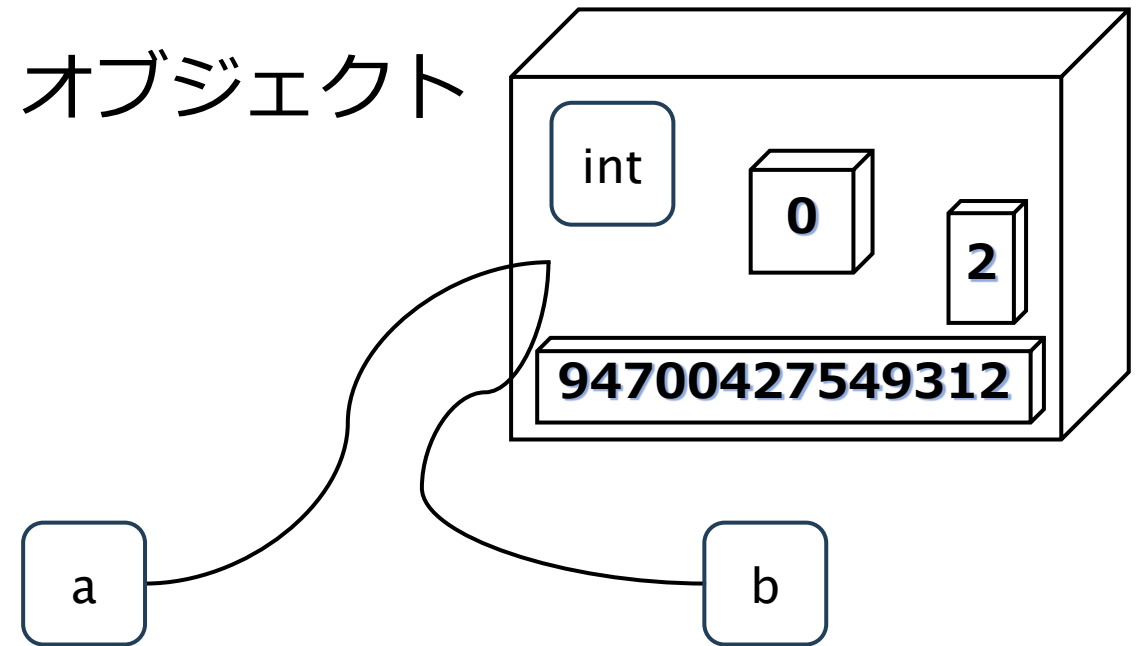
Python対話型インタプリタで変数宣言と値代入の挙動を確認

1. aという名前に値0を代入する。
すると、整数値0を入れたオブジェクトが作られる。
2. aの値を出力する。
3. bにaの値を代入し、0があるオブジェクトの箱にラベルbも貼る。
4. bの値を出力する。

※以降「>>>」の右はPython対話型インタプリタで書いたコードを示す。その記号がない行は命令の実行結果を示す。記号横のスペースは自動挿入。プログラマが手動追加する必要はない。左端の数字は命令の番号。実際には表示されない。

```
1 >>> a = 0
2 >>> a      ※「=」は代入の意味で、等しい
0              の意味ではない。
3 >>> b = a
4 >>> b
0
```

オブジェクト



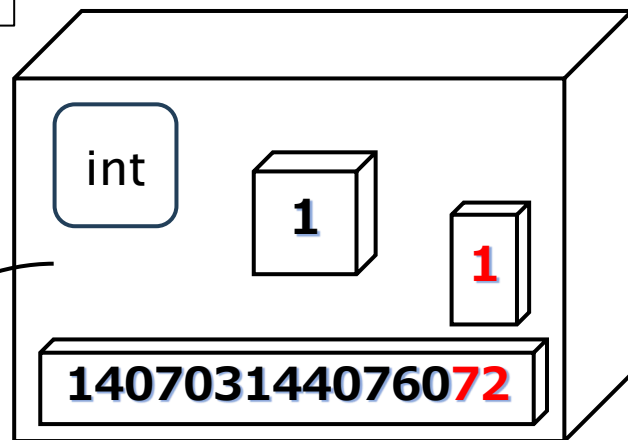
aに値を再代入してaとbの値を出力

```
1 >>> a = 0
2 >>> a
0
3 >>> b = a
4 >>> b
0
5 >>> a = 1
6 >>> a
1
7 >>> b
0
```

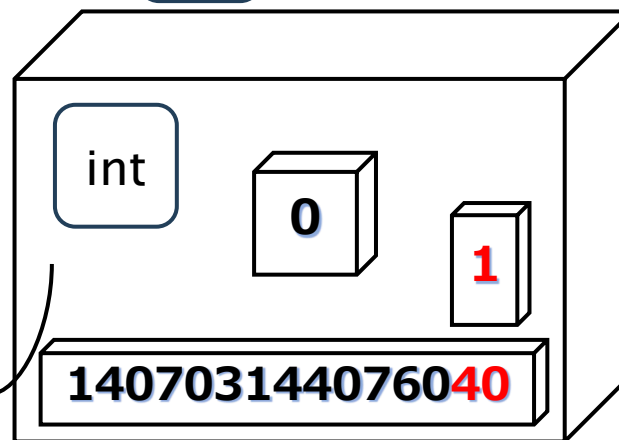
再代入

一度代入した変数に再度新しい値を代入すると、参照先が新しい値に更新される。

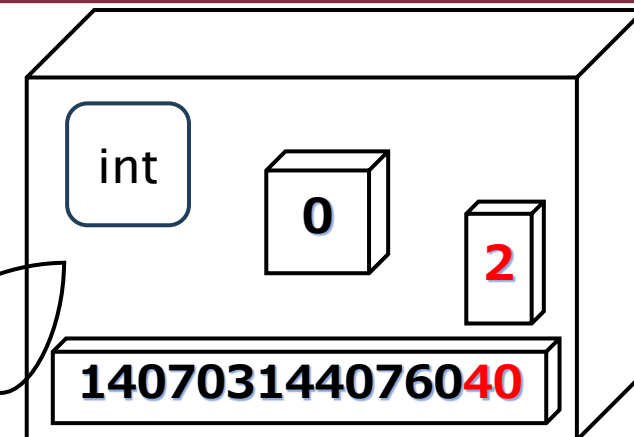
a



a



b



b

リテラル

```
1 >>> a = 0
2 >>> a
0
3 >>> b = a
4 >>> b
0
5 >>> a = 1
6 >>> a
1
7 >>> b
0
8 >>> 'a'
'a'
9 >>> True
True
```

リテラル (Literal) は「文字どおり」「字義どおり」を意味する語。プログラム内でデータの値が直接的に記述された表現。

数値リテラル (Numeric Literal) : 数値データを表現するためのリテラル。整数や浮動小数点数などの数値を直接表現。例 42 や 3.14 は数値リテラル。

文字列リテラル (String Literal) : テキストや文字列を表現するためのリテラル。通常、ダブルクォーテーション ("") またはシングルクォーテーション (' ') で囲まれる。例 "Hello, World!" は文字列リテラル。

ブーリアンリテラル (Boolean Literal) : 真偽値 (真または偽) を表現するためのリテラル。Python では、**True (真)** または **False (偽)** と表現し、これらの 2 値のみ。他にもリテラルの種類はあるがここでは割愛。

変数の命名規則

- 変数名として利用できるのは次の文字。
 - 小文字の英字 (aからzまで)
 - 大文字の英字 (AからZまで)
 - 数字 (0から9まで)
 - アンダースコア (_)
- 大文字と小文字は区別。 : `thing`, `Thing`, `THING`は全部異なる。
- 変数名の先頭は英字かアンダースコア (_) であること。
 - 良い例: `data_1`, `_name`
 - 悪い例: `123`, `1data`
 - 変数名の先頭がアンダースコアの名前はPythonにおいて特別な意味を持つものとして扱われる。
- Pythonの予約語 (キーワードとも呼ぶ) は変数名として利用できない。

Pythonの予約語

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with

- プログラミング言語の語彙として予め約束されている表現であり、上記はPythonにおける予約語の例。
- プログラミング言語によって予約語は異なる。
- プログラムで確認も可能。

```
1 import keyword
2 print( keyword.kwlist )
```

名前付けの記法

- 名前から変数の内容が推察できる。
 - 良い例: `title`, `data`
 - 微妙例: `x`, `y` (一概に`x,y`が悪いわけではないので場合によっては使用するのも可)
- 長すぎず、短すぎない。
 - 良い例: `user`, `year`
 - 微妙例: `login_user_name`, `birth_of_the_year`
 - 極端に短い `x`, `y` は上述の通り, 場合によっては可
- 見た目が似ている名前は避ける。(大文字、小文字による区別も該当)
 - 例: `user` / `usr`, `temp` / `tmp`, `Year` / `year`
- 決められた記法で統一する
 - 良い例: `UserName` / `MyData` / `BirthDate`
 - 微妙例: `UserName` / `User_Name`
- 英単語にする
 - 良い例: `name`, `result`
 - 微妙例: `namae`, `kekka`
 - 日本語名にするのはお勧めしない

名前付けの記法

- どのように名前を付ければよいか？
- CamelCase記法
 - 先頭文字は小文字，単語の区切りを大文字にする
 - 例: `userName`
- Pascal記法
 - 先頭文字も含め，全ての単語の頭文字を大文字にする
 - 例: `UserName`
- アンダースコア記法
 - 全ての文字を小文字にし，単語の区切りはアンダースコア `_` にする
 - 例: `user_name`

名前付けのルールを強制するものはない。しかし、統一的な名前付けのルールは各自でプログラミングする際に設けておくと、プログラムが整然とするので無難。

まとめ

- OS標準の**ターミナル系ソフト**と**テキストエディタ**を併用することは、プログラムを作成するための**基本的かつ汎用的な方法**である。
- **基本を身につければ**制限の強い環境下でプログラミングを遂行できたり、言語や作業内容が変更しても**応用が効いたりする**。
- **Hello, world!プログラム**とは、プログラミングを初めて学ぶ際やソフトの開発時にプログラミング環境が正常かを確認する際などに作る**習慣**。
- **データ型（型）**とは**値を意味づける** オブジェクト **分類情報**である。
- Pythonにおける**変数**とはオブジェクトの在処を示す**参照情報**である。
- 代入は**オブジェクトと変数の関連づけ**。
- Pythonは**対話型インタプリタ**を持ち、**簡易的なプログラミングが可能**。

