

# プログラミング言語

## 第2回 4月15日

担当：篠沢 佳久  
栗原 聡

2019年度：春学期

# 講義のホームページ

- ◆ 講義に関する情報
  - <http://lecture.comp.ae.keio.ac.jp/program2019/>

# まずは注意点

- ◆ ドキュメントに「Python」というフォルダを作って、作成したプログラムはそこに保存するようにすること
- ◆ （第一回の講義資料を参照）
- ◆ 日吉ITCのPCでは、ドキュメントの中のPythonフォルダは、"Z:¥Documents¥Python" を指す

# 日吉ITCの場合 (OSはWindows10)

The image shows a Windows 10 desktop with a blue background. On the left, the Start menu is open, displaying various application tiles. A red box highlights the 'エクスプローラー' (File Explorer) tile. A large yellow arrow points from this tile to a File Explorer window on the right. The File Explorer window shows the 'クイック アクセス' (Quick access) view, with a red box highlighting the 'ドキュメント' (Documents) folder under 'よく使用するフォルダー' (Frequently used folders).

① Windowsボタン→  
エクスプローラー

② 「ドキュメント」をクリック

# 本日の内容

- ◆ 対話型シェルの使い方
- ◆ データの型
- ◆ 演算子
- ◆ 練習問題

# 対話型シェル (インタラクティブシェル)

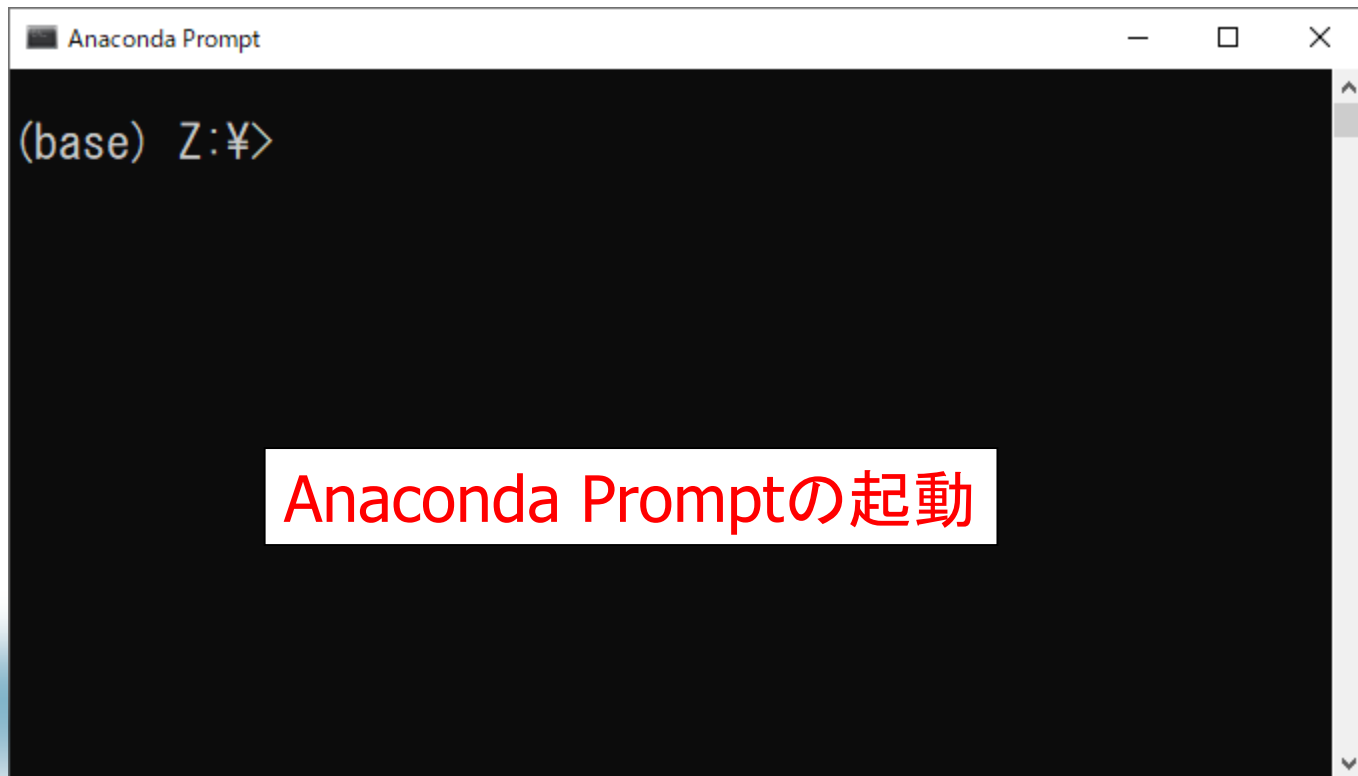
対話型シェルの起動と終了  
シェル上でのPythonプログラムの実行方法

# まずは: 対話型シェル

- ◆ プログラムは、多数の文で構成されている
- ◆ しかし、中には、実行したいことが一文で書けてしまうこともある
- ◆ Pythonには、この「一文プログラム」の実行ができるツールが提供されている
  - 実は、複数文に渡っても実行できる、優れたもの
- ◆ それが、対話型シェル(インタラクティブシェル)
- ◆ 一文ごとにプログラムの実行ができる

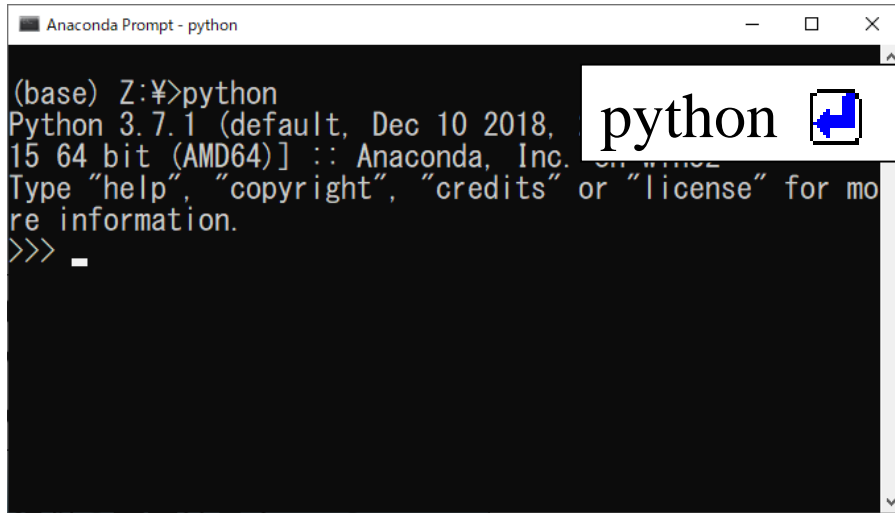
# 対話型シェルの起動①

- ◆ 「Windowsボタン」→「Anaconda3(64-bit)」  
→「Anaconda Prompt」



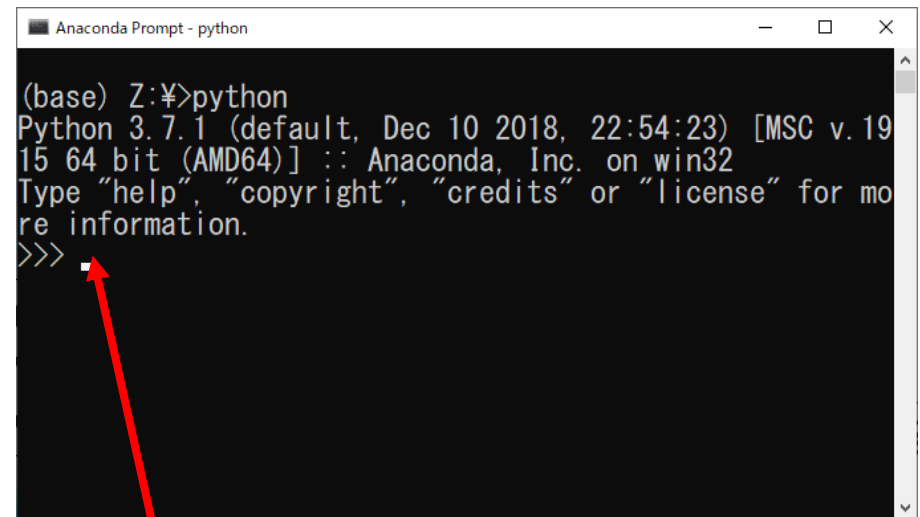


# 対話型シェルの起動②



```
Anaconda Prompt - python
(base) Z:\>python
Python 3.7.1 (default, Dec 10 2018, 15:15:15) [AMD64] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license()" for more
>>>
```

python と入力



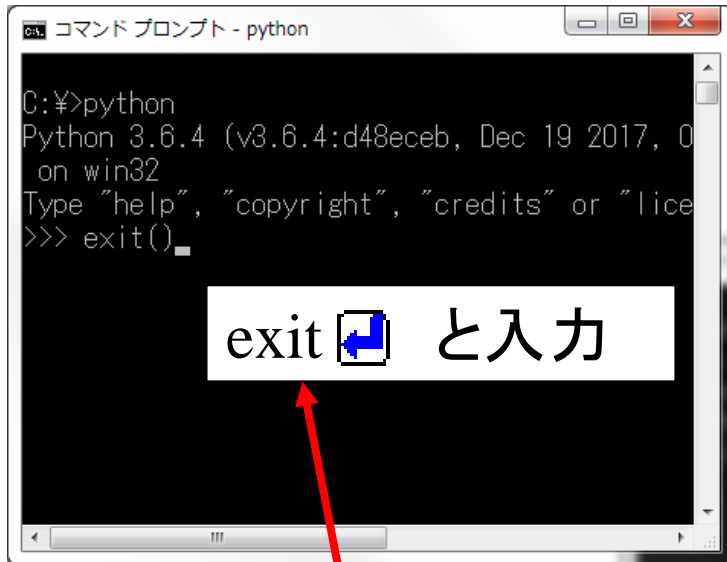
```
Anaconda Prompt - python
(base) Z:\>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license()" for more
>>>
```

このプロンプトが表示されている時, Pythonのプログラムが実行可能

プロンプトが「>>>」と変わること  
に注目\*

\*講義資料の画面は日吉ITCのPC  
と異なる場合があります

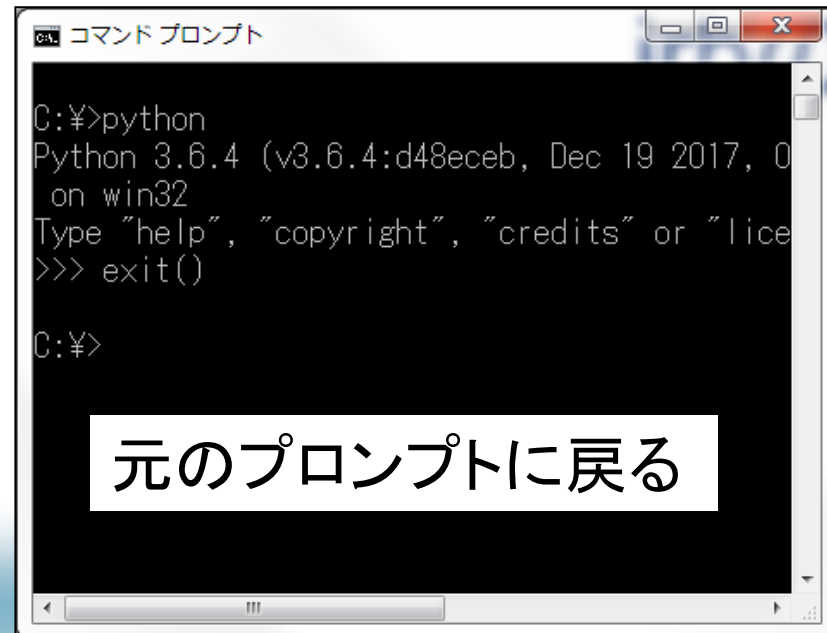
# 対話型シェルの終了



```
C:\>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 0
on win32
Type "help", "copyright", "credits" or "lice
>>> exit()
```

exit  と入力

もしくはCtrl+Z  
(Ctrlを押しながらZ)

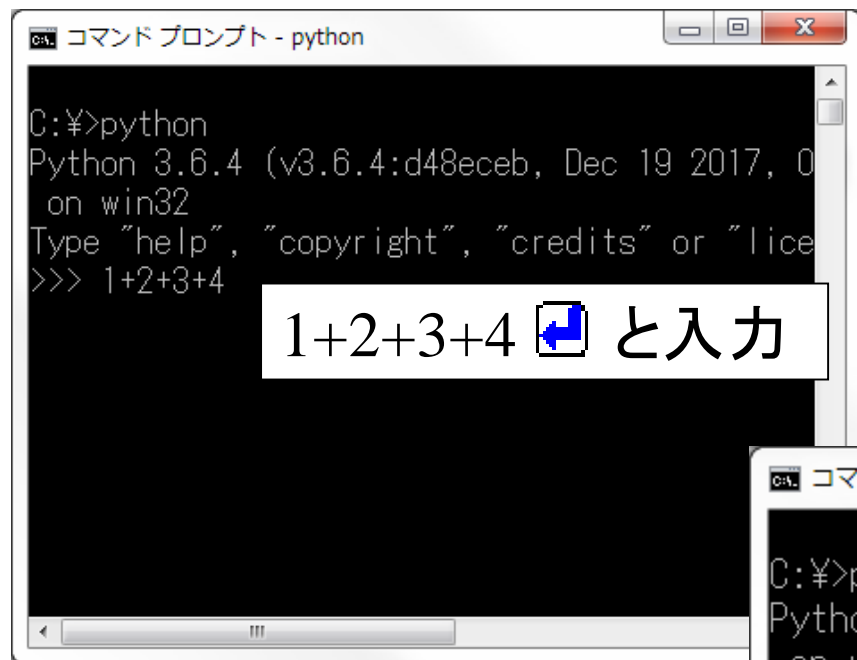


```
C:\>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 0
on win32
Type "help", "copyright", "credits" or "lice
>>> exit()

C:\>
```

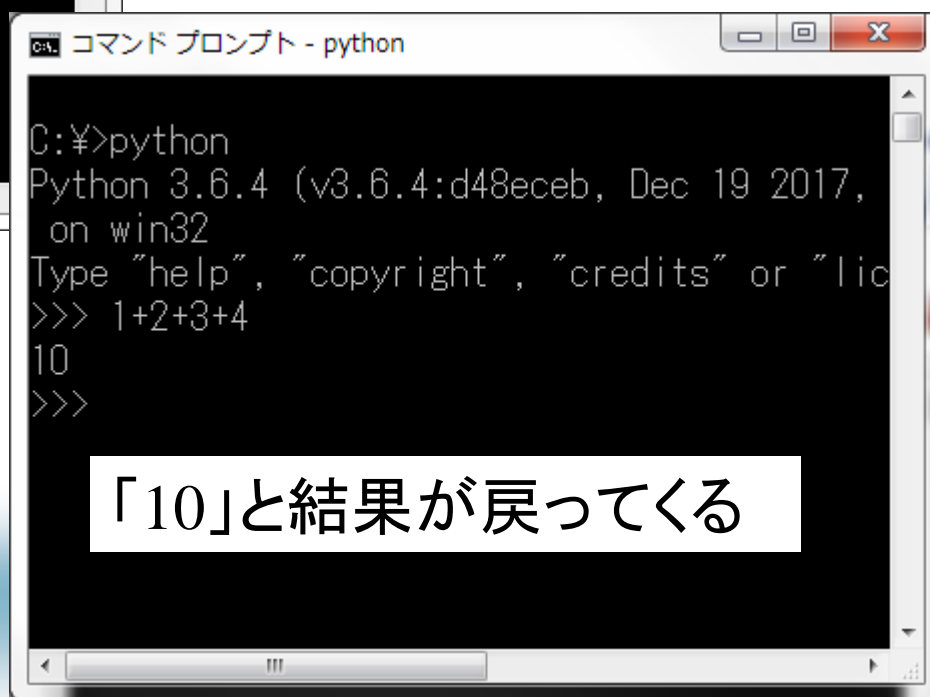
元のプロンプトに戻る

# 対話型シェル上での入力方法①



```
C:\>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 0
on win32
Type "help", "copyright", "credits" or "lice
>>> 1+2+3+4
```

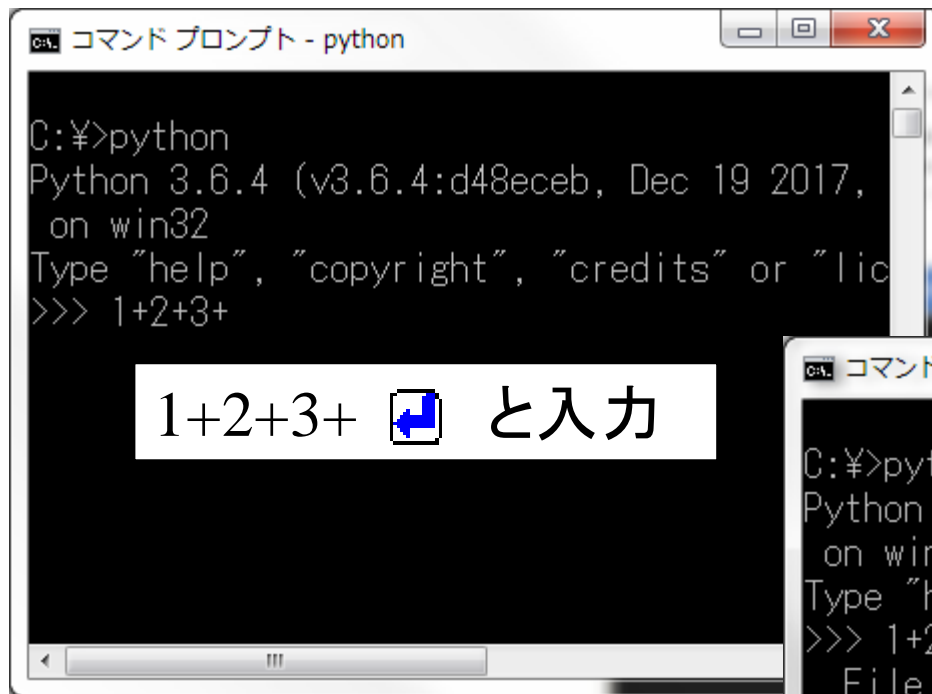
1+2+3+4 と入力



```
C:\>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017,
on win32
Type "help", "copyright", "credits" or "lic
>>> 1+2+3+4
10
>>>
```

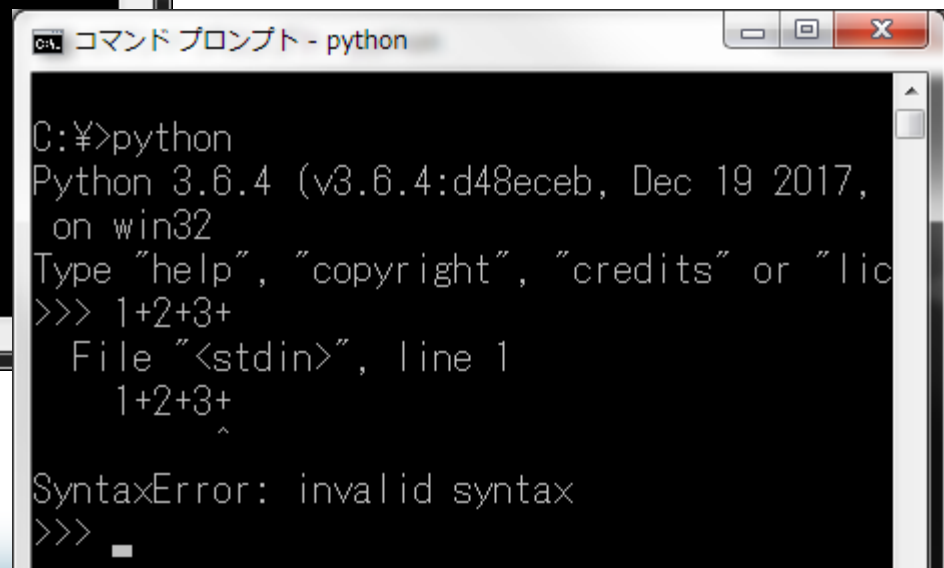
「10」と結果が戻ってくる

# 対話型シェル上での入力方法②



```
C:\>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017,
on win32
Type "help", "copyright", "credits" or "lic
>>> 1+2+3+
```

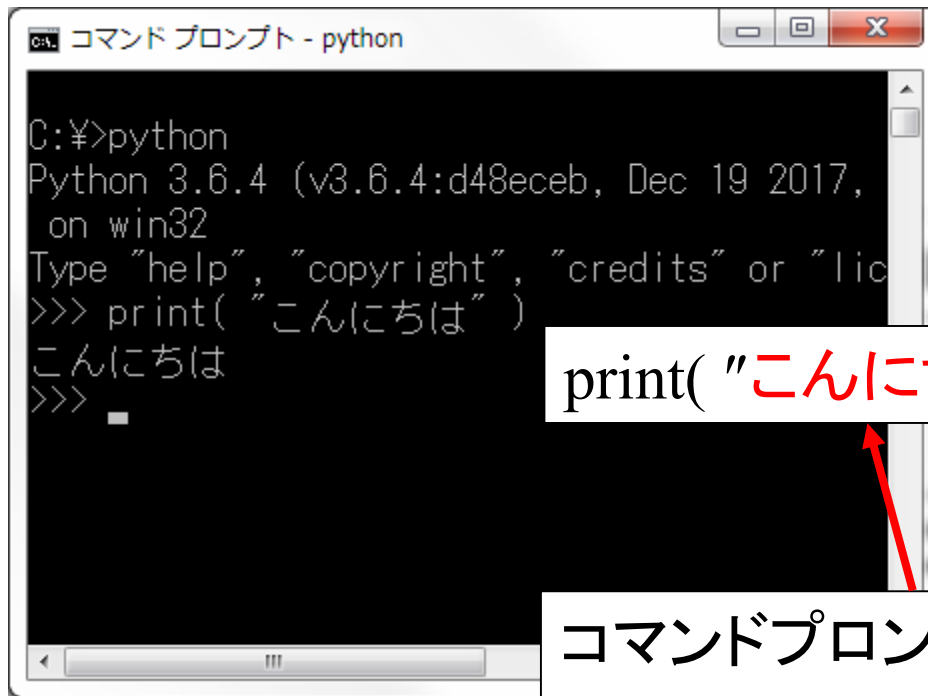
1+2+3+ と入力



```
C:\>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017,
on win32
Type "help", "copyright", "credits" or "lic
>>> 1+2+3+
File "<stdin>", line 1
  1+2+3+
    ^
SyntaxError: invalid syntax
>>> .
```

エラーメッセージ (SyntaxError) が表示される  
→ 式として誤りがあることを知らせてくれる

# 対話型シェル上での入力方法③



```
C:\>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017,
on win32
Type "help", "copyright", "credits" or "lic
>>> print( "こんにちは" )
こんにちは
>>> _
```

print( "こんにちは" ) と入力

コマンドプロンプト上での日本語入力  
「半角/全角」でローマ字変換



# 対話型シェル上での実行例

- ◆ 電卓がわりにも使える
- ◆ 日本語以外は半角文字で入力する
- ◆ 入力した後は, Enterキーを入力すると結果が戻ってくる

代入式

標準出力

```
>>> 1+2+3+4
10
>>> 2*3*4*5*6*7*8*9*10
3628800
>>> x=2.0
>>> print(x)
2.0
>>> import math
>>> math.sqrt(x)
1.4142135623730951
>>> math.pi
3.141592653589793
>>> math.sin(math.pi/4)
0.7071067811865475
>>> math.sqrt(2)/2
0.7071067811865476
```

数学用関数を使う場合  
**import math**と入力しておく

数学用関数

平方根 `math.sqrt()`

$\pi$  `math.pi`

sin `math.sin()`

# 算術演算子

演算子	用途	例	演算結果
+	加減	3+2	5
-	減算	4-2	2
*	乗算	2*2	4
/	除算	4/2	2.0
//	除算	4//2	2
%	剰余	5%2	1
**	冪	5**3	125

「/」は小数点以下を切り捨てない 「//」は小数点以下を切り捨てる

# 演算の優先順位①

- ◆  $5 + 10 * 5$   
55

- ◆  $(5 + 10) * 5$   
75

- ◆  $10 * 2 + 3$   
23

- ◆  $10 * (2 + 3)$   
50

- ◆  $3 * 4 ** 2$   
48

- ◆  $(3 * 4) ** 2$   
144

優先される

空白は空ける必要はありません。スライドを見やすくするためにつけています

**\*\***  
**/ \* %**  
**+ -**

4\*\*2を行ない、その結果を3倍する

(3\*4)を行ない、その結果を2乗する



# 括弧の書き方

```
>>> (3+6)
```

```
9
```

```
>>> ((3+6))
```

```
9
```

```
>>> (((3+6)))
```

```
9
```

```
>>> ((3+6)
```

```
.. )
```

```
9
```

```
>>> (3+6))
```

```
File "<stdin>", line 1
```

```
(3+6))
```

```
^
```

```
SyntaxError: invalid syntax
```

複数個の()を用いてもよい

必ず閉じる

左右の個数が合っていない場合は  
エラー(SyntaxError)が表示される

## 演算の優先順位②

- ◆  $5 + - 3$   
2
- ◆  $-(3 - 10)$   
7
- ◆  $3^{**} - 2$   
0.111111111111111111
- ◆  $- 3^{**} 2$   
- 9
- ◆  $- 3^{**} - 2$   
-0.111111111111111111
- ◆  $(- 3)^{**} - 2$   
 $\Rightarrow 0.111111111111111111$

優先される

括弧  $()$

$**$

符号  $+ -$

$/ * \%$

$+ -$

3 \*\* -2を行ない, その結果をマイナスとする

# 数学用関数①

## ◆ 平方根

- `math.sqrt( 2 )`

使用する場合, `import math`  
を忘れないこと

## ◆ 三角関数

- `math.sin( math.pi )`
- `math.cos( math.pi )`
- `math.tan( math.pi )`

`sin  $\pi$`

`cos  $\pi$`

`tan  $\pi$`

`$\pi$`

`math.pi`

単位はラジアン

# 数学用関数②

- ◆ 自然対数

- `math.log( math.e )`

`e`

`math.e`

- ◆ 常用対数

- `math.log10( 100 )`

- ◆ 指数

- `math.exp( 2 )`

その他の数学関数

<https://docs.python.jp/3/library/math.html>

# 数学用関数③

使用する場合, import math  
を忘れないこと

```
>>> import math
>>> math.e
2.718281828459045
>>> math.log(math.e)
1.0
>>> math.log10(100)
2.0
>>> math.exp(2)
7.38905609893065
```

# 代入文①

- ◆  $x = 2$
- ◆  $y = 10$
- ◆  $(x + y) * 2$   
24
- ◆  $x * y$   
20
- ◆  $\text{math.sqrt}(y / x)$   
2.23606797749979

$x, y$  を変数

$x = 2$  を代入文と呼ぶ

## 代入文②

- ◆  $x = 2$
- ◆  $y = 10$
- ◆  $a = (x + y) / 2$
- ◆  $b = x * y$
- ◆  $c = y / x$
- ◆  $d = \text{math.sqrt}(c)$

前ページと同じ計算

求めた値を別の変数(a, b, c, d)に代入することも可能

# 変数の値の表示

```
>>> x=2
>>> y=10
>>> a=(x+y)/2
>>> b=x*y
>>> c=y/x
>>> d=math.sqrt(c)
>>> print(a)
6.0
>>> print(b)
20
>>> print(c)
5.0
>>> print(d)
2.23606797749979
>>>
```

変数の値を表示するためにはprint文を用いる

print( 変数名 )



# 変数の値の表示

```
>>> x=2
>>> y=10
>>> a=(x+y)/2
>>> b=x*y
>>> c=y/x
>>> d=math.sqrt(c)
>>> a
6.0
>>> b
20
>>> c
5.0
>>> d
2.23606797749979
>>>
```

対話型シェル上では、変数名のみでも値は表示できますが、`print`を用いて下さい

## 代入文③

```
>>> x=2
>>> y=10
>>> a=(x+y+z)/2
Traceback (most recent call last):
  File "<pysHELL#63>", line 1, in <module>
    a=(x+y+z)/2
NameError: name 'z' is not defined
```

エラーメッセージ  
→ 変数zは未定義

```
x=2
y=10
a=(x+y+z)/2
→ 変数 z は未定義
```

## 代入文③'

```
>>> x=2
>>> y=10
>>> z=0
>>> a=(x+y+z)/2
>>> print(a)
6.0
```

z=0と定義しておく

代入文の右側に現れる変数はそれ以前に定義しなければならない

# エラー出力①

- ◆ 式をPythonの文法通りに記述しなかった場合、エラーが出力されます

```
>>> w
Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    w
NameError: name 'w' is not defined
>>> math.Pi
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    math.Pi
AttributeError: module 'math' has no attribute 'Pi'
```

変数wは未定義

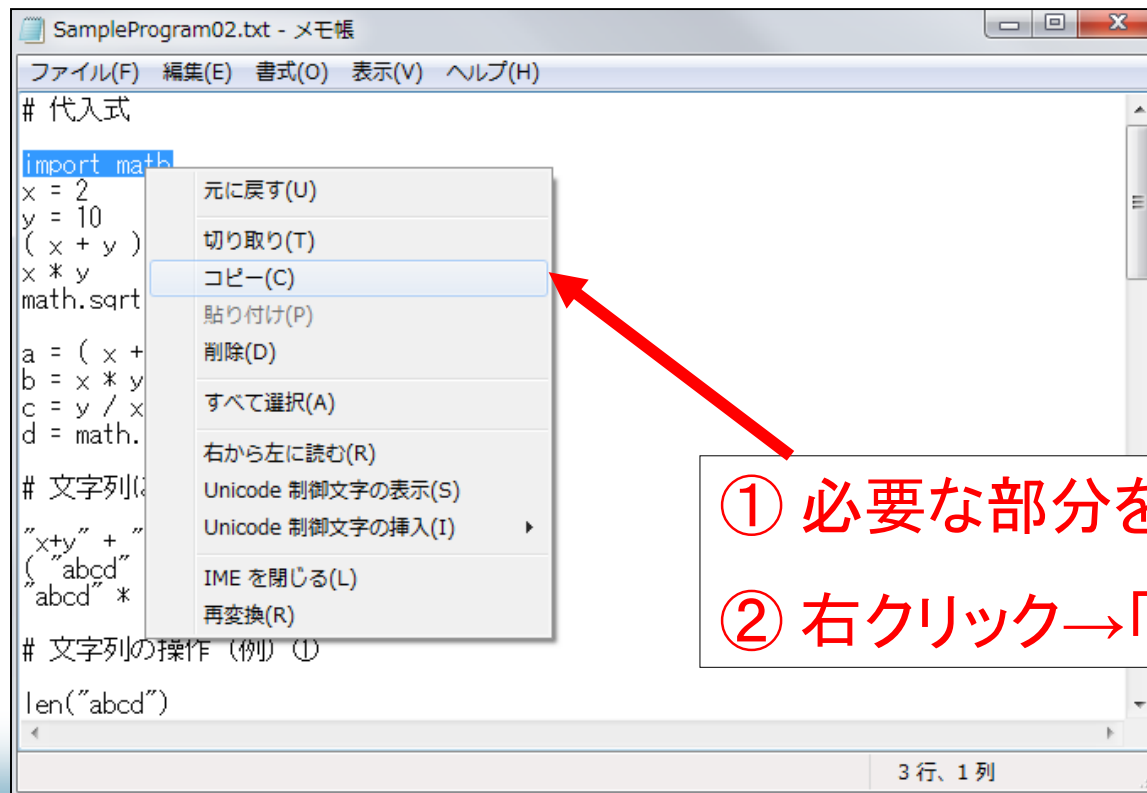
math.Piは存在しない→正しくはmath.pi

## エラー出力②

- ◆ 式をPythonの文法通りに記述しなかった場合, エラーが出力されます
  - エラーの原因が書かれています
  - まずはどこにエラーがあるか見つけて下さい
  - 次になぜ間違えたのか, どう訂正すればよいかを考えて下さい

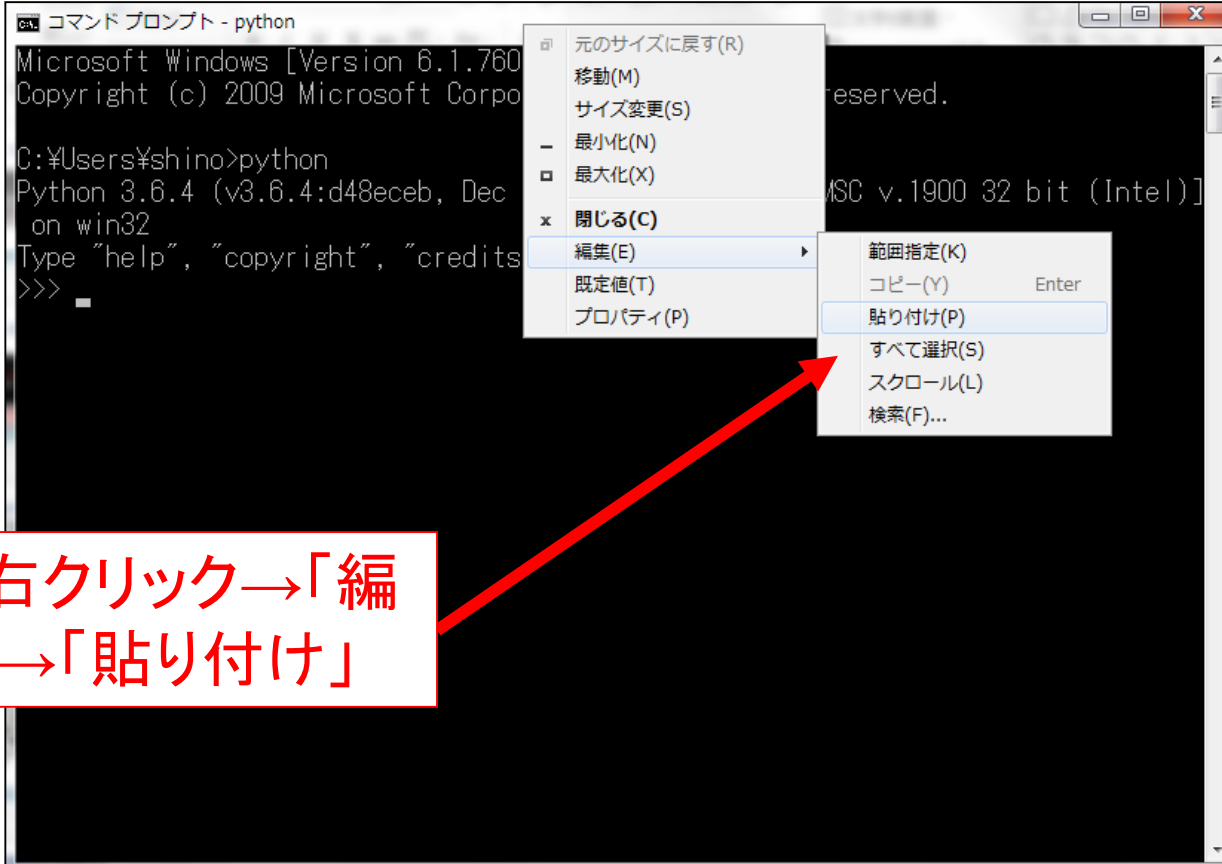
# 対話型シェルへの貼り付け①

テキストエディタで書いたプログラムをシェル上で動かしたい



- ① 必要な部分をドラッグ
- ② 右クリック→「コピー」

## 対話型シェルへの貼り付け②

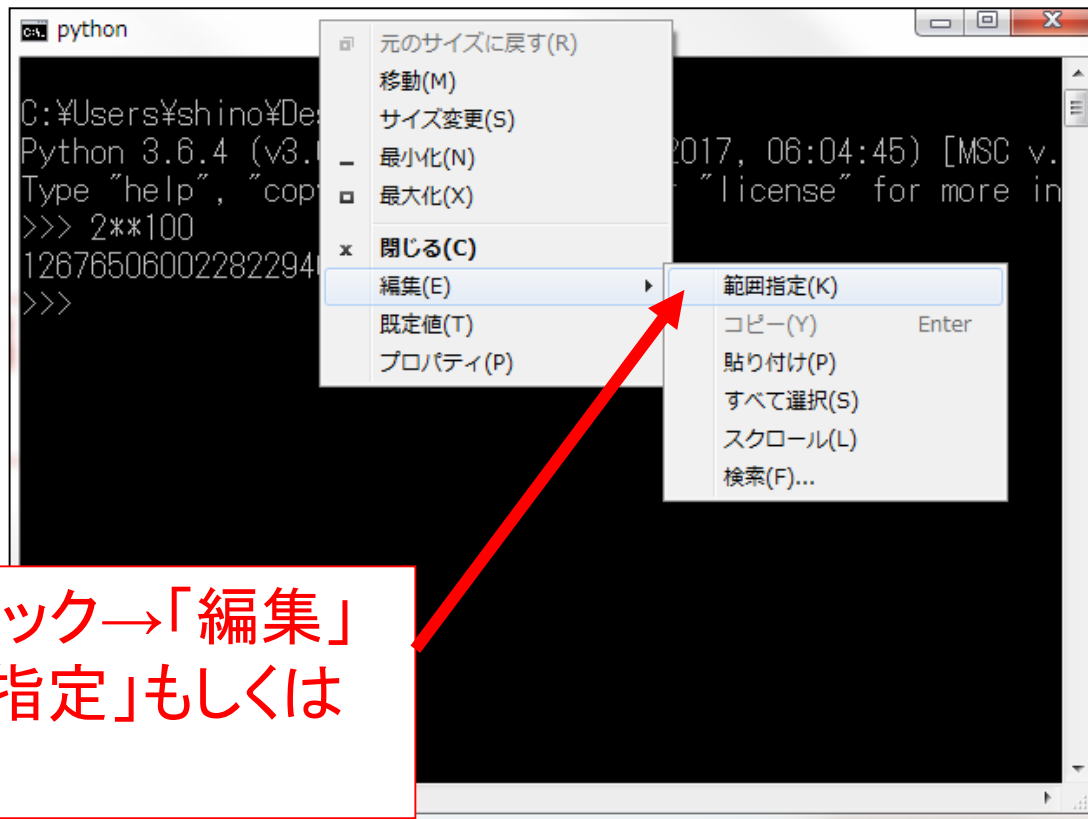


③ 右クリック→「編集」→「貼り付け」

複数行のコピー&ペーストもできますが、一行ごとに実行して下さい

# 対話型シェルからのコピー①

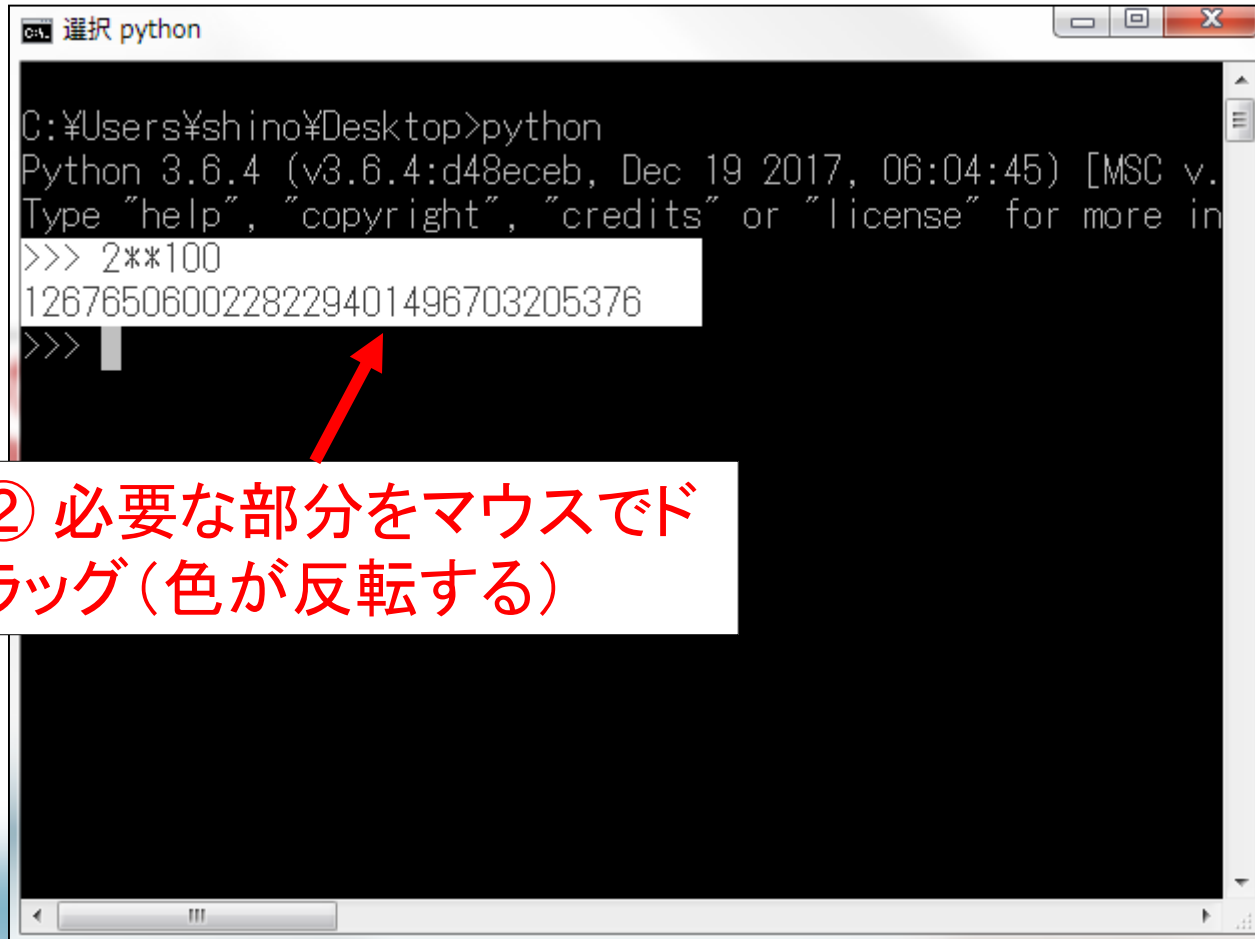
シェル上のプログラム，結果をメモ帳などにコピーしたい場合



① 右クリック→「編集」  
→「範囲指定」もしくは  
「マーク」



## 対話型シェルからのコピー②



```
C:\Users\shino\Desktop>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.
Type "help", "copyright", "credits" or "license" for more in
>>> 2**100
1267650600228229401496703205376
>>>
```

② 必要な部分をマウスでドラッグ(色が反転する)

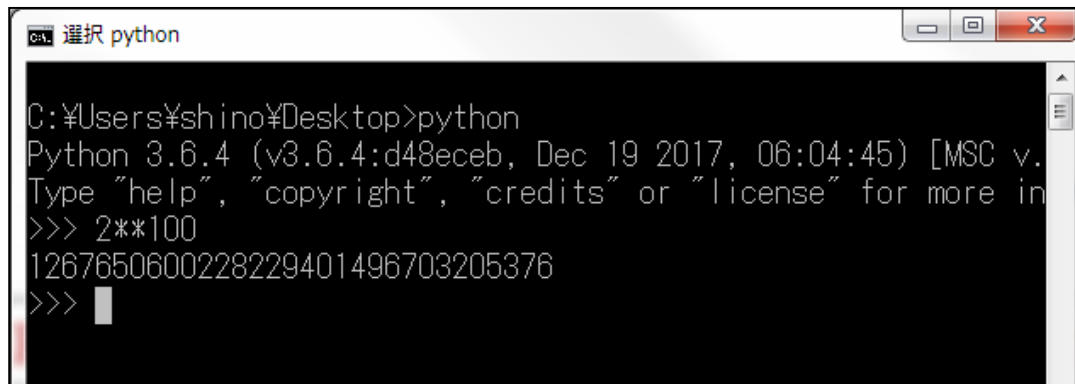
# 対話型シェルからのコピー③

③ 右クリック→「編集」→「コピー」

④ エディタ, MS-Word上で  
右クリック→「貼り付け」

# 対話型シェルの画面のコピー①

対話型シェルの画面をMS-Word等に貼り付けたい場合



```
C:\Users\shino\Desktop>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.
Type "help", "copyright", "credits" or "license" for more in
>>> 2**100
1267650600228229401496703205376
>>> █
```

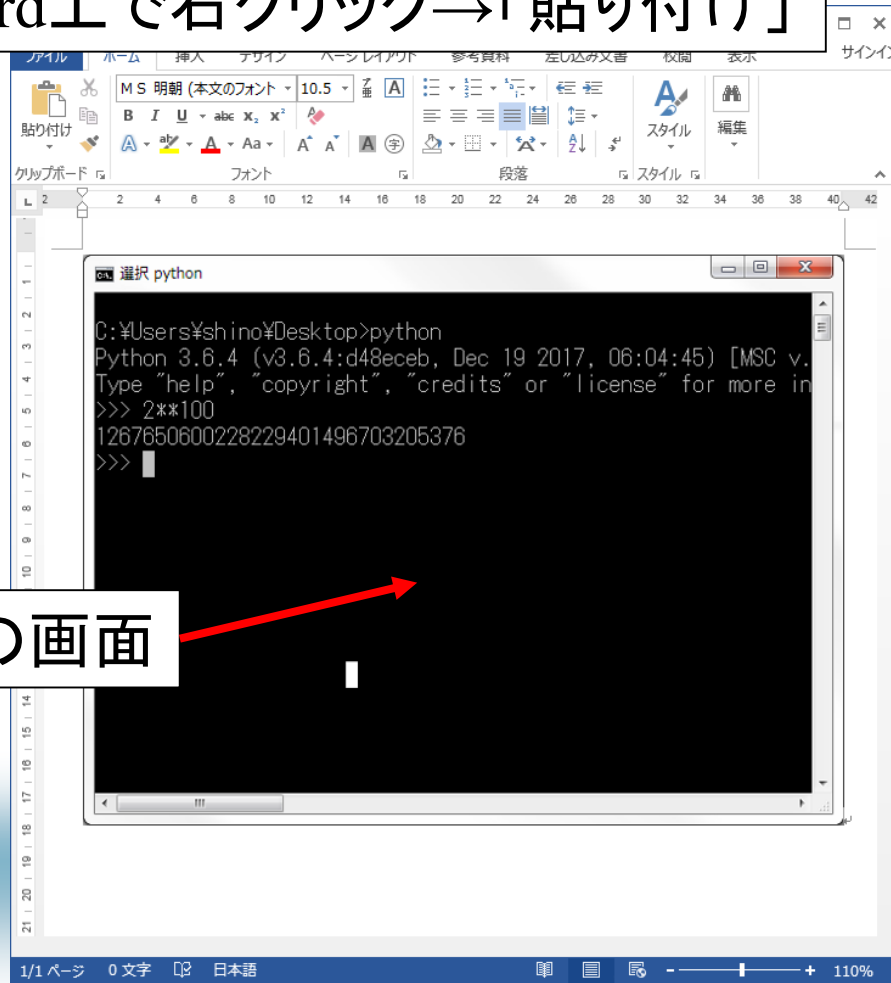
① 対話型シェルのウィンドウをクリック

② Alt キーを押しながら PrintScrn



# 対話型シェルの画面のコピー②

## ③ MS-Word上で右クリック→「貼り付け」



実行結果の画面

# データの型

整数型

浮動小数点型

文字列型

# 数値には型がある①

- ◆ さて、次のようになる理由を考えてみよう

```
>>> 3
```

```
3
```

```
>>> 3.0
```

```
3.0
```

```
>>> 3+2
```

```
5
```

```
>>> 3+2.0
```

```
5.0
```

```
>>> 3.0+2.0
```

```
5.0
```

ヒント:

小数点がある数と小数点がない数に違いがある

## 数値には型がある②

- ◆ 「3」は「整数」
- ◆ 「3.0」は「小数」
- ◆ 「3+2」は「整数と整数を足し算」  
→ 結果は「整数」
- ◆ 「3+2.0」は「整数と小数を足し算」  
→ 結果は「小数」
- ◆ 「3.0+2.0」は「小数と小数を足し算」  
→ 結果は「小数」

# データの型①

- ◆ データ
  - コンピュータの演算・操作の対象
  - 文字列, 小数点のある数, 小数点のない数
- ◆ データの型
  - そのデータに適用が許される演算・操作の集合
- ◆ 小数点のない数: 小数点のない数による加減乗除
- ◆ 小数点のある数: 小数点のある数による加減乗除
  - 小数点のない数に小数点のある数を足そうとすると(それはできない), 前者を小数点のある数に変換して, 足し算をする
  - この変換を「**型変換**」という



# 型変換

- ◆ 「3+2.0」
  - 「整数と小数を足し算」することはできない
  - 整数「3」を小数「3.0」に型変換
  - 「3.0+2.0」として「小数と小数を足し算」
  - 結果は小数「5.0」

# データの型②

- ◆ Pythonにある主なデータ型:
  - 小数点のない数: 整数
    - integer
  - 小数点のある数: 浮動小数点数(小数)
    - float(floating point number)
  - 文字列
    - string
  - 論理型(True/False)
    - boolean

# データには型がある(例)

```
>>> 2+3  
5
```

```
>>> 2+3.0  
5.0
```

```
>>> 2.0+3  
5.0
```

```
>>> 2.0+3.0  
5.0
```

```
>>> 3/2  
1.5
```

```
>>> 3//2  
1
```

```
>>> 2.0//3  
0.0
```

```
>>> 3//2.0  
1.0
```

除算(/)の結果は小数点  
以下を切り捨てない

除算(//)の結果は小数点  
以下を切り捨てる

整数と小数を計算すると整数は小数に自動的に  
変換され、結果は小数となる

# 型変換

- ◆ 「2.0//3」
  - 「小数を整数で割り算」することはできない
  - 整数「3」を小数「3.0」に型変換
  - 「2.0//3.0」として「小数を小数で割り算」
  - 0.666666...を小数点以下切捨て
  - 結果は小数「0.0」

# 型変換

- ◆ 「3//2.0」
  - 「整数を小数で割り算」することはできない
  - 整数「3」を小数「3.0」に型変換
  - 「3.0//2.0」として「小数を小数で割り算」
  - 1.5を小数点以下切捨て
  - 結果は小数「1.0」

# データには型がある(例)

```
>>> 10e5
```

```
1000000.0
```

```
>>> 10e-5
```

```
0.0001
```

```
>>> 10**5
```

```
100000
```

```
>>> 10**-5
```

```
1e-05
```

```
>>> 10.0**-5
```

```
1e-05
```

10e5

→ 10 × 10.0<sup>5</sup>

表示の違いに注目

# データには型がある(例)

小数型

```
>>> 10e5  
1000000.0
```

```
>>> 10e-5  
0.0001
```

```
>>> 10**5  
100000
```

```
>>> 10**-5  
1e-05
```

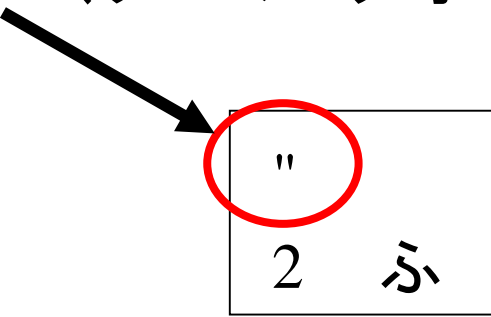
```
>>> 10.0**-5  
1e-05
```

整数型

# 文字列型

- 文字列を使用する場合は" "（ダブルクォート）で囲む

```
>>> "abcd"  
'abcd'  
>>> "xyz"  
'xyz'  
>>> "123.4"  
'123.4'  
>>> "3+3"  
'3+3'
```



"  
2 ふ

小数も" "で  
囲むと文字列

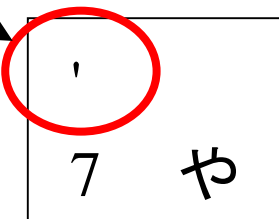
式も" "で囲  
むと文字列



# 文字列型

- ◆ もしくは, 文字列を使用する場合は' ' (シングルクォート) で囲む

```
>>> 'abcd'
'abcd'
>>> 'xyz'
'xyz'
>>> '123.4'
'123.4'
>>> '3+3'
'3+3'
```



# ダブルクォートとシングルクォートの違い

- ◆ 基本, 同じです
- ◆ ダブルクォートで挟んだ中にダブルクォートは使えません
- ◆ `"java ruby c++ "python""` (×)
- ◆ その場合, シングルダブルクォートで挟みます
- ◆ `'java ruby c++ "python"'` (○)

# トリプルクォートで囲んだ文字列

```
>>> """abc
... def
... ghi
... jkl
... """
...
'abc¥ndef¥nghi¥njl¥n'
```

複数行になる場合  
、トリプルクォート  
( """ で囲む )

改行コード (¥n) が入る

# 文字列型の演算子

- ◆ +
  - 文字列 + 文字列
  - 二つの文字列の連結
- ◆ \*
  - 文字列 \* 整数
  - 整数 \* 文字列
  - 文字列を整数回繰り返す

# 文字列にも演算が可能①

```
>>> "abcd"*2
'abcdabcd'
>>> "abcd"+"efgh"
'abcdefgh'
>>> "Abc"*3
'AbcAbcAbc'
```

「+」「\*」は可能

```
>>> "abcd"-"cd"
```

Traceback (most recent call last):

File "<pyshell#112>", line 1, in <module>

"abcd"-"cd"

TypeError: unsupported operand type(s) for -: 'str' and 'str'

引き算は不可能

実行できないプログラムを入力した  
場合エラーメッセージが返ってくる

## 文字列にも演算が可能②

```
>>> "x+y"+"z"
```

x+y は数式ではなく文字列

```
'x+yz'
```

```
>>> ("abcd"+"efg")*2
```

括弧も可能

```
'abcdefgabcdefg'
```

```
>>> "abcd"*2+"efg"
```

「\*」の方が優先順位は強い

```
'abcdabcdefg'
```

# 文字列の操作①

- ◆ `len(文字列)`
  - 文字列の長さを求める(値は整数)
- ◆ 文字列`[n]`
  - 文字列のn番目の文字を取り出す
  - ただし先頭の文字を0番目とする
- ◆ 文字列`[ a : b ]`
  - 文字列のa番目からb-1番目までの文字列を取り出す
  - ただし先頭の文字を0番目とする

# 文字列の操作②

- ◆ 文字列[ **a :** ]
  - 文字列のa番目から最後まで文字列を取り出す
  - ただし先頭の文字を0番目とする
- ◆ 文字列[ **:a** ]
  - 先頭から文字列のa-1番目までの文字列を取り出す
  - ただし先頭の文字を0番目とする
- ◆ 文字列[ **::-1** ]
  - 文字列を反転



# 文字列の操作③

- ◆ 文字列1.**find**( 文字列2 )
  - 文字列1の中から文字列2の位置を調べる
  - ただし先頭の文字を0番目とする
  - 文字列2が存在しない場合は-1
- ◆ 文字列.**replace**( 変換前の文字, 変換後の文字 )
  - 文字列の中から変換前の文字を変換後の文字に置換する

# 文字列の操作④

- ◆ 文字列.**upper()**
  - 文字列中の小文字を大文字に変換
- ◆ 文字列.**lower()**
  - 文字列中の大文字を小文字に変換

# 文字列の操作(例)①

"abcd"

0	1	2	3
a	b	c	d

先頭の a は文字列中で0番目の文字として扱われる

0番目から0番目の文字  
→ a

1番目から2番目の文字  
→ bc

```
>>> len("abcd")
4
>>> "abcd"[0]
'a'
>>> "abcd"[1]
'b'
>>> "abcd"[0:1]
'a'
>>> "abcd"[1:3]
'bc'
>>> "abcd"[1:]
'bcd'
>>> "abcd"[:3]
'abc'
```

1番目以降の文字  
→ bcd

2番目までの文字  
→ abc

## 文字列の操作(例)②

```
>>> len("python programming")
```

```
18
```

```
>>> "python programming"[ 2 ]
```

```
't'
```

```
>>> "python programming"[ 5: 10 ]
```

```
'n pro'
```

```
>>> "python programming"[::-1]
```

文字列を反転

```
'gnimmargorp nohtyp'
```

```
>>> "python programming".find("pro")
```

"pro"の位置を検索

```
7
```

```
>>> "python programming".replace("p","P")
```

```
'Python Programming'
```

"p"(小文字)を"P"(大文字)に置換

# 文字列の先頭的位置

"python programming"

0	1	2	3	4	5	6	7	8	9	10	11	12	13
p	y	t	h	o	n		p	r	o	g	r	a	m

14	15	16	17
m	i	n	g

"python programming"[ 5: 10 ]  
=> "n pro"

5番目から9番目の文字列

## 文字列の操作(例)③

```
>>> "abcd".upper()
'ABCD'
>>> "ABCD".lower()
'abcd'
>>> "abCD".upper()
'ABCD'
>>> "abCD".lower()
'abcd'
```

文字列.upper()  
大文字に変換

文字列.lower()  
小文字に変換

その他の文字列操作

<https://docs.python.jp/3/library/stdtypes.html?highlight=str#str>

## 文字列の操作(例)④

```
>>> "abcd"*2
```

```
'abcdabcd'
```

```
>>> "abcd"*"2"
```

文字列同士の「\*」演算は不可能

```
Traceback (most recent call last):
```

```
File "<pyshell#141>", line 1, in <module>
```

```
"abcd"*"2"
```

```
TypeError: can't multiply sequence by non-int of type 'str'
```

```
>>> len("abcd"*2)
```

len("abcdabcd")

```
8
```

```
>>> len("abcd"[0:3]*2)
```

```
6
```

"abcd"[0:3]→"abc", "abc"\*2→"abccabc"→len("abccabc")

# 型により不可能な演算①

```
>>> "abcd"-"ab" ← 文字列同士の引き算は不可能
```

```
Traceback (most recent call last):
```

```
File "<pyshell#142>", line 1, in <module>  
    "abcd"-"ab"
```

```
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

```
>>> len(2) ← 整数の長さは求めることができない
```

```
Traceback (most recent call last):
```

```
File "<pyshell#143>", line 1, in <module>  
    len(2)
```

```
TypeError: object of type 'int' has no len()
```

```
>>> len("2") ← 文字列の長さは求めることができる
```

```
1
```



## 型により不可能な演算②

```
>>> 1+"abcd"
```

整数と文字列の足し算は不可能

```
Traceback (most recent call last):
```

```
File "<pyshell#145>", line 1, in <module>
```

```
1+"abcd"
```

```
TypeError: unsupported operand type(s) for +:
```

```
'int' and 'str'
```

```
>>> 1+1.0
```

整数と小数の足し算は可能

```
2.0
```

```
>>> 1+"3"
```

整数と文字列の足し算は不可能

```
Traceback (most recent call last):
```

```
File "<pyshell#147>", line 1, in <module>
```

```
1+"3"
```

```
TypeError: unsupported operand type(s) for +:
```

```
'int' and 'str'
```

```
>>> "1"+"3"
```

文字列と文字列の足し算は可能

```
'13'
```

# データの型のまとめ①

- ◆ type(値)
  - 型が表示

```
>>> type(3)
```

```
<class 'int'>
```

int(整数型)

```
>>> type(3.1415)
```

```
<class 'float'>
```

float(小数型)

```
>>> type("3.1415")
```

```
<class 'str'>
```

str(文字列型)

## データの型のまとめ②

```
>>> a=3  
>>> type(a)  
<class 'int'>
```

int(整数型)

```
>>> a=3.141  
>>> type(a)  
<class 'float'>
```

float(小数型)

```
>>> a="3.141"  
>>> type(a)  
<class 'str'>
```

str(文字列型)

# 演算子

算術演算子  
比較演算子

# 整数型の算術演算子

演算子	用途	例	演算結果
+	加減	3+2	5
-	減算	4-2	2
*	乗算	2*2	4
/	除算	4/2	2.0
//	除算	4//2	2
%	剰余	5%2	1
**	冪	5**3	125

「/」は小数点以下を切り捨てない 「//」は小数点以下を切り捨てる

# 浮動小数点数型の算術演算子

演算子	用途	例	演算結果
+	加減	3.1+2.2	5.3
-	減算	4.2-2.1	2.1
*	乗算	2.1*2.1	4.41
/	除算	4.5/3.0	1.5
//	除算	4.5//3.0	1.0
%	剰余	5.0%2.1	0.8
**	冪	2.1**0.5	1.44913

# 比較演算子

演算子	用途	例	演算結果
==	等	3==2	False
>	大	4 > 2	True
<	小	4 < 2	False
>=	大 or 等	4>=2	True
<=	小 or 等	4<=2	False
!=	非等	3!=2	True
in	含まれる	"x" in "xyz"	True

# 比較演算子①

- 比較も演算です

```
>>> 3 == 2
False
>>> 4 > 2
True
>>> 4 < 2
False
>>> 4 >= 2
True
>>> 4 <= 2
False
>>> 3 != 1
True
```

これは等号  
等しいか否かを判定する演算子

演算式が正しい場合  
→「True」を返す

演算式が正しくない場合  
→「False」を返す



# 比較演算子②

整数と小数の比較は可能

```
>>> 2 == 2
True
>>> 2 < 3
True
>>> 2 >= 3
False
>>> 2 != 3
True
>>> "ab" == "ab"
True
>>> "ab" == "abc"
False
>>> "ab" < "abc"
True
```

文字列の比較も可能

```
>>> 2.1 < 2.2
True
>>> 2.1 > 2.2
False
>>> 2.0 == 2
True
```

```
>>> 2 < "2"
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in
<module>
  2<"2"
TypeError: '<' not supported between
instances of 'int' and 'str'
>>> 2.1 == 2.10
True
```

整数と文字列の比較  
は不可能

## 比較演算子③

```
>>> 10*2 == 20  
True
```

```
>>> (5+4) < 20  
True
```

```
>>> (2+3) == 8  
False
```

```
>>> (2+3) == (10-5)  
True
```

```
>>> (2+3) != (10-5)  
False
```

```
>>> len("abcd") < 4  
False
```

```
>>> "abcd".upper() == "ABCD"  
True
```

```
>>> True == True  
True  
>>> True == False  
False
```

式と式との比較  
も可能

len("abcd")にて  
整数4となる

## 比較演算子④

in演算子  
abcが含まれるのでTrue

```
>>> "abc" in "xyzabcdefg"  
True  
>>> "ABC" in "xyzabcdefg"  
False  
>>> "ABC" in "xyzabcdefg".upper()  
True  
>>> ("ABC" in "xyzabcdefg".upper()) == True  
True
```

True

## 少々不思議な結果①

```
0.9
>>> 0.99
0.99
>>> 0.999
0.999
>>> 0.9999
0.9999
>>> 0.99999
0.99999
>>> 0.999999
0.999999
>>> 0.9999999
0.9999999
>>> 0.99999999
0.99999999
>>> 0.999999999
0.999999999
>>> 0.9999999999
0.9999999999
>>> 0.99999999999
0.99999999999
```

[illegible]

# 有限桁数かつ四捨五入の世界だからです

## 少々不思議な結果②

```
>>> 2/3
0.6666666666666666
>>> 2/3 == 0.6666666666
False
>>> 2/3 == 0.6666666666666666666666666666
True
```

勿論，有限桁数かつ四捨五入の世界だからです

# 型変換

# 型変換①

- ◆ 整数と小数の演算
  - 整数は小数に自動的に変換され、結果は小数となる
- ◆ 小数と文字列の演算
  - 不可能
- ◆ 他のデータ型に変換することを型変換と呼ぶ

# 整数型への変換①

- ◆ 整数に変換

`int( 3.1415 )`

`int( "3" )`

`int( "3" ) + 5`

整数へ変換

`int(値)`



## 整数型への変換②

```
>>> int( 3.1415 )
```

```
3
```

```
>>> int( "3" )
```

```
3
```

```
>>> "3" + 5
```

文字列と整数の足し算は不可能

```
Traceback (most recent call last):
```

```
File "<pyshell#56>", line 1, in <module>
```

```
"3" + 5
```

```
TypeError: must be str, not int
```

```
>>> int( "3" ) + 5
```

```
8
```

文字列を整数に変換することで可能

# 小数型への変換①

- ◆ 小数に変換

`float( 3 )`

`float( "3.1415" )`

`float( "3" )`

`float( "3.1415" ) * 2.5`

小数へ変換

`float( 値 )`

## 小数型への変換②

```
>>> float(3)
3.0
>>> float( "3.1415" )
3.1415
>>> float( "3" )
3.0
>>> float( "3.1415" ) * 2.5
7.8537500000000001
```

文字列を小数に変換することで可能

# 文字列型への変換①

- ◆ 文字列に変換

`str( 3 )`

`str( 3.1415 )`

`str( 3 ) + "5"`

`str( 3.1415 ) * 2`

文字列へ変換

`str( 値 )`

## 文字列型への変換②

```
>>> str( 3 )
```

```
'3'
```

```
>>> str( 3.1415 )
```

```
'3.1415'
```

```
>>> str( 3 ) + "5"
```

```
'35'
```

```
>>> str( 3.1415 ) * 2
```

```
'3.14153.1415'
```

# 型変換のまとめ

```
>>> float( "3.1415" )
```

小数に変換

```
3.1415
```

```
>>> int( "3.1415" )
```

(注意) 整数に一回で変換できない

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
ValueError: invalid literal for int() with base 10: '3.1415'
```

```
>>> int( float( "3.1415" ) )
```

小数→整数に変換

```
3
```

```
>>> float( int( float( "3.1415" ) ) )
```

小数→整数→小数に変換

```
3.0
```

```
>>> str( int( float( "3.1415" ) ) )
```

小数→整数→文字列に変換

```
'3'
```

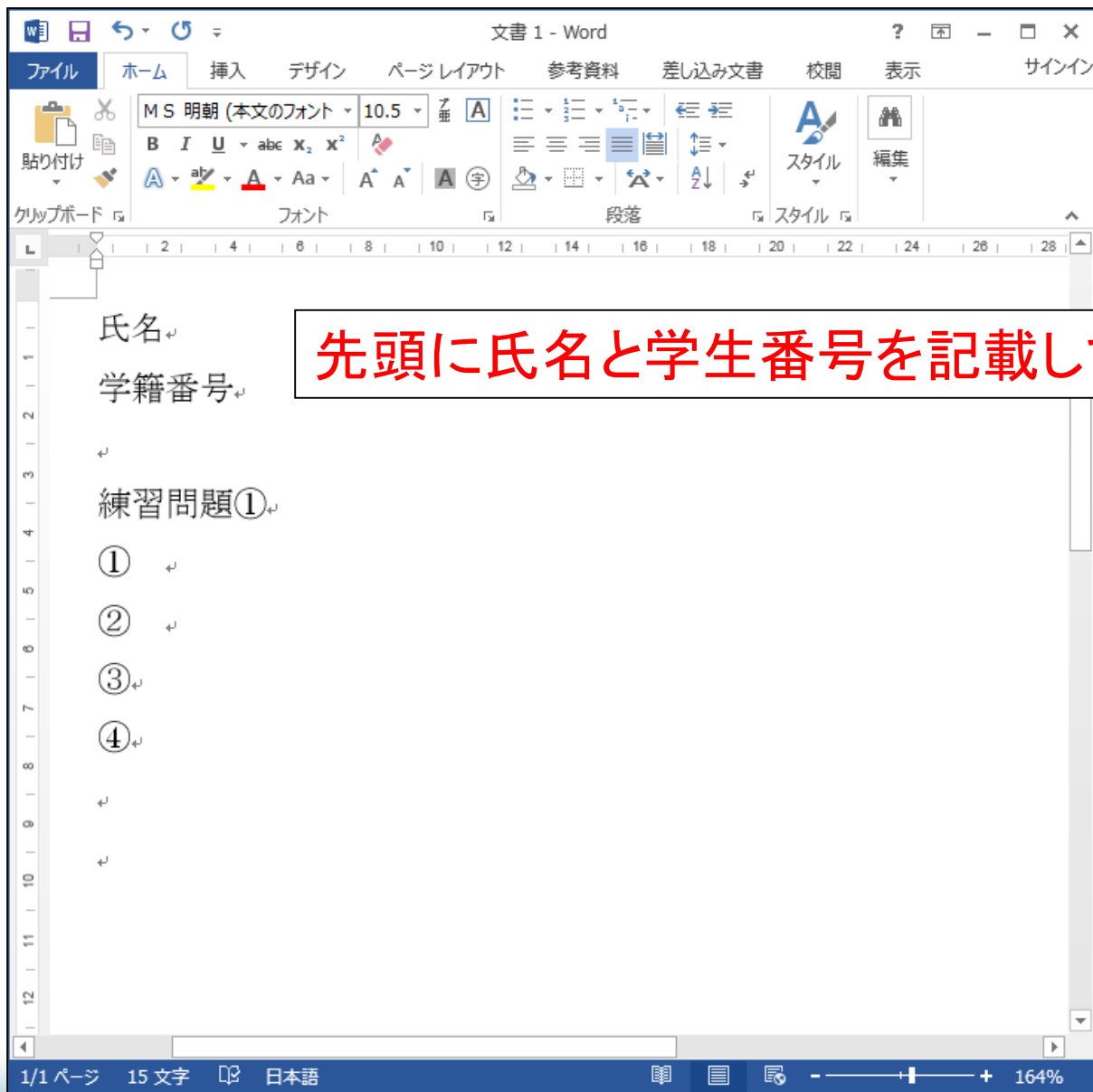
# 練習問題

# 練習問題

- ◆ 次頁以降の練習問題を行ないなさい.
- ◆ MS-Wordに回答を記述し, keio.jpに提出して下さい
  - 提出先
    - 第二回講義練習問題
  - 提出締め切り
    - 4/22(月)10時30分まで\*
  - 提出するMS-Wordファイルの先頭に学籍番号, 氏名を書いて下さい

\*通常, 講義中の練習問題の提出締め切りは, 講義当日の16:30までですが, 履修登録の関係上, 今回は一週間先とします





先頭に氏名と学生番号を記載して下さい

ファイル名は自由につけて下さい

# 練習問題

1. 練習①～⑤の実行結果がどうなるか答えなさい(実行する前に, まず答えを考えて下さい. その答えが正しいかどうかを確認するため, その後実行することが望ましい)
2. 結果が True , False になる式を5個ずつ考えなさい

# 練習①(データには型がある) 結果がどうなるか考えて下さい

①  $1+2+3+4+5$

- 結果は？

③  $(1+2+3+4+5) // 2$

- 結果は？

②  $1+2+3+4+5.0$

- 結果は？

④  $(1+2+3+4+5) // 2.0$

- 結果は？

「//」は除算の結果を小数点以下, 切り捨てる

# 練習②(整数と小数)

## 結果がどうなるか考えて下さい

①  $4/3$

「/」は除算の結果を小数点以下  
、切り捨てない

②  $(4/3)*3$

③  $4//3$

④  $(4//3)*3$

⑤  $(4.0//3)*3$

## 練習③整数型への変換 結果がどうなるか考えて下さい

- ①  $3.1415 * 2$
- ②  $\text{int}( 3.1415 ) * 2$
- ③  $\text{int}( 3.1415 ) * 2.0$
- ④  $\text{int}( 3.1415 * 2 )$
- ⑤  $\text{int}( 3.1415 / 2 )$

# 練習④小数型への変換

## 結果がどうなるか考えて下さい

①  $3 * 2$

②  $\text{float}(3) * 2$

③  $\text{int}(3.1415) + \text{float}(2)$

④  $(\text{int}(3.1415) + \text{float}(2)) * \text{int}(3.1415)$

⑤  $3.1415 * \text{int}(3.1415) / 3.1415$

## 練習⑤文字列型への変換 結果がどうなるか考えて下さい

- ① `str(3)*2`
- ② `len(str(3.1415))`
- ③ `str(int(3.1415))`
- ④ `str( "abc" > "abcd" )`

## 練習⑥(型変換)

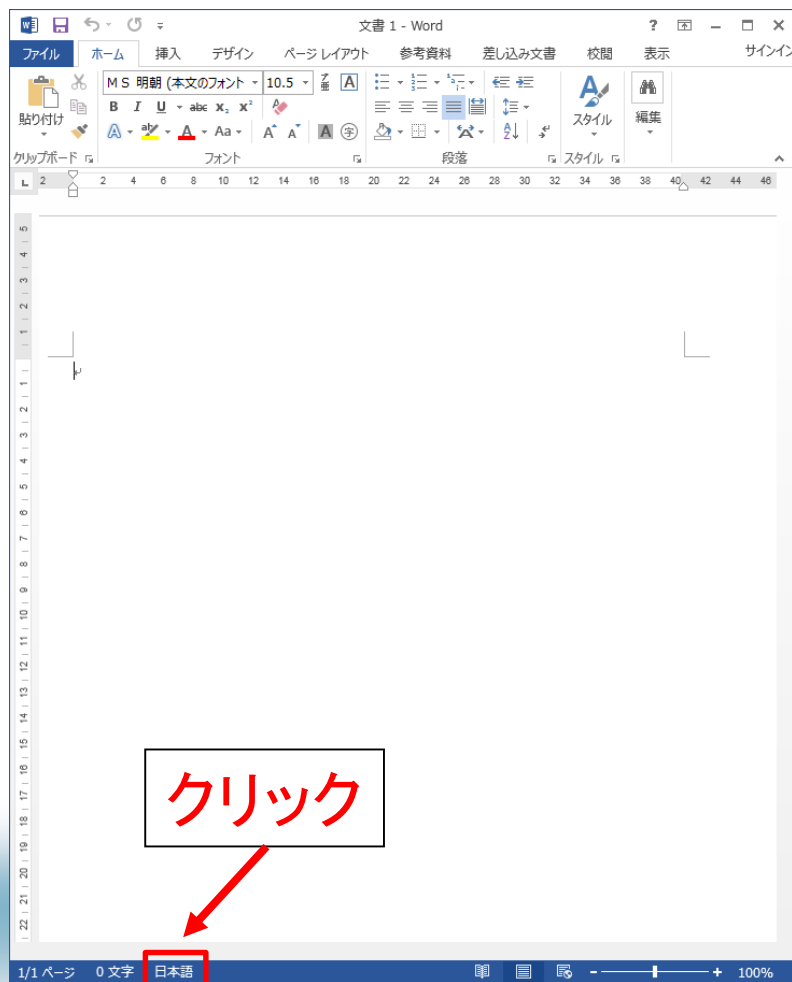
- ①  $1 / 2 + 3 / 2$  を行なうと、結果は 2.0 となります。結果を型変換(int)によって 1 にするためには、式のどこを変えればよいか？
- ②  $(1 / 3 + 2 / 3) / 3$  を行なうと、結果は 0.333333333 となります。結果を型変換(int)によって 0 にするためには、式のどこを変えればよいか？



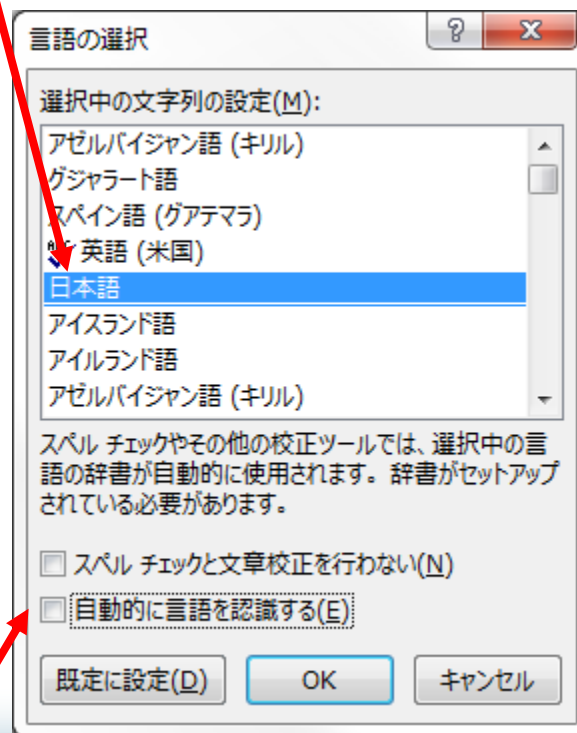
# MS-Wordにおける注意

他言語に変換されてしまう場合

# MS-Wordで日本語以外の文字が表示される場合



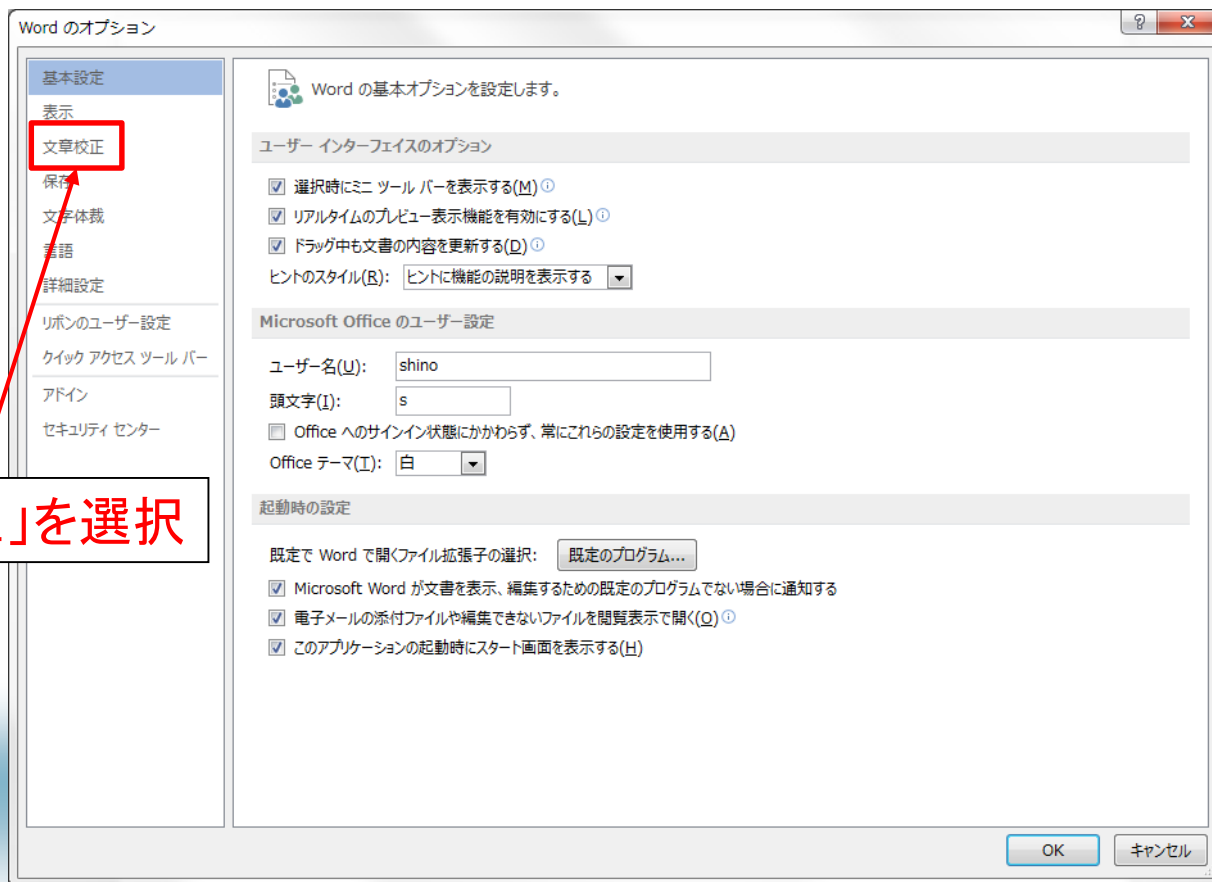
「日本語」になっているか



「自動的に言語を認識する」  
のチェックを外す

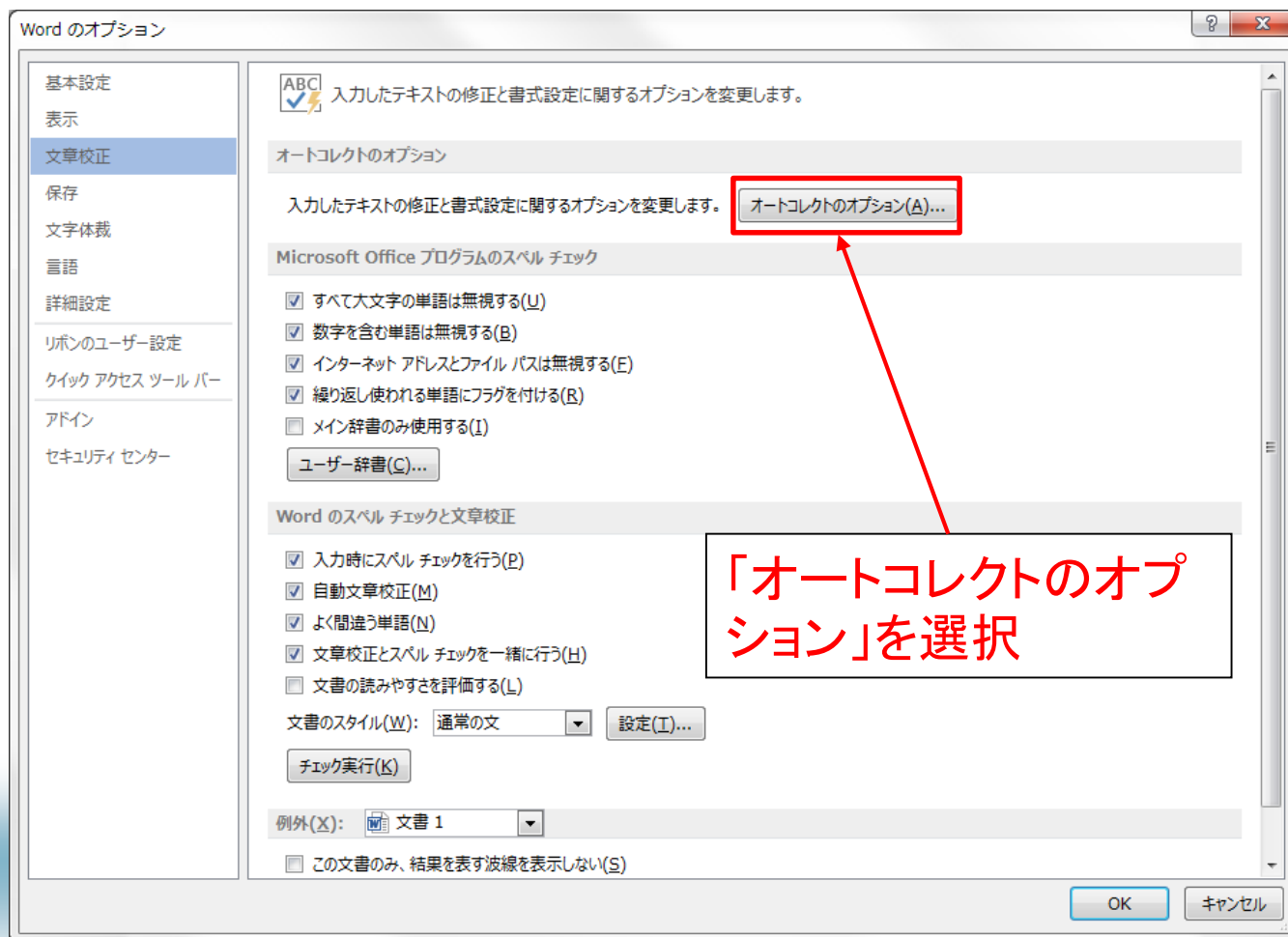
# MS-Wordでダブルクオートが自動変換される場合①

「ファイル」→「オプション」→「文章校正」



「文章校正」を選択

# MS-Wordでダブルクオートが自動変換される場合②



# MS-Wordでダブルクオートが自動変換される場合③

「入力オートフォーマット」を選択

オートコレクト

オートコレクト 数式オートコレクト **入力オートフォーマット** オートフォーマット 操作

入力中に自動で変更する項目

<input type="checkbox"/> 左右の区別がない引用符を、区別がある引用符に変更する	<input checked="" type="checkbox"/> 序数 (1st, 2nd, 3rd, ...) を上付き文字に変更する
<input type="checkbox"/> 分数 (1/2, 1/4, 3/4) を分数文字 (組み文字) に変更する	<input checked="" type="checkbox"/> ハイフンをダッシュに変更する
<input type="checkbox"/> '*'、'_' で囲んだ文字列を '太字'、'斜体' に書式設定する	<input checked="" type="checkbox"/> 長音とダッシュを正しく使い分ける
<input checked="" type="checkbox"/> インターネットとネットワークのアドレスをハイパーリンクに変更する	
<input type="checkbox"/> 行の始まりのスペースを字下げに変更する	

入力中に自動で書式設定する項目

<input checked="" type="checkbox"/> 箇条書き (行頭文字)	<input checked="" type="checkbox"/> 箇条書き (段落番号)
<input checked="" type="checkbox"/> 罫線	<input checked="" type="checkbox"/> 表
<input type="checkbox"/> 既定の見出しスタイル	<input type="checkbox"/> 日付スタイル
<input checked="" type="checkbox"/> 結語のスタイル	

入力中に自動で行う処理

<input checked="" type="checkbox"/> リストの始まりの書式を前のリストと同じにする
<input checked="" type="checkbox"/> Tab/Space/BackSpace キーでインデントとタブの設定を変更する
<input type="checkbox"/> 設定した書式を新規スタイルとして登録する
<input checked="" type="checkbox"/> かっこを正しく組み合わせる
<input type="checkbox"/> 日本語と英数字の間の不要なスペースを削除する
<input checked="" type="checkbox"/> '記' などに対応する '以上' を挿入する
<input checked="" type="checkbox"/> 頭語に対応する結語を挿入する

OK キャンセル

チェックを外す

「OK」をクリック