プログラミングの基礎　Python 操作説明編

１．Python の利用について

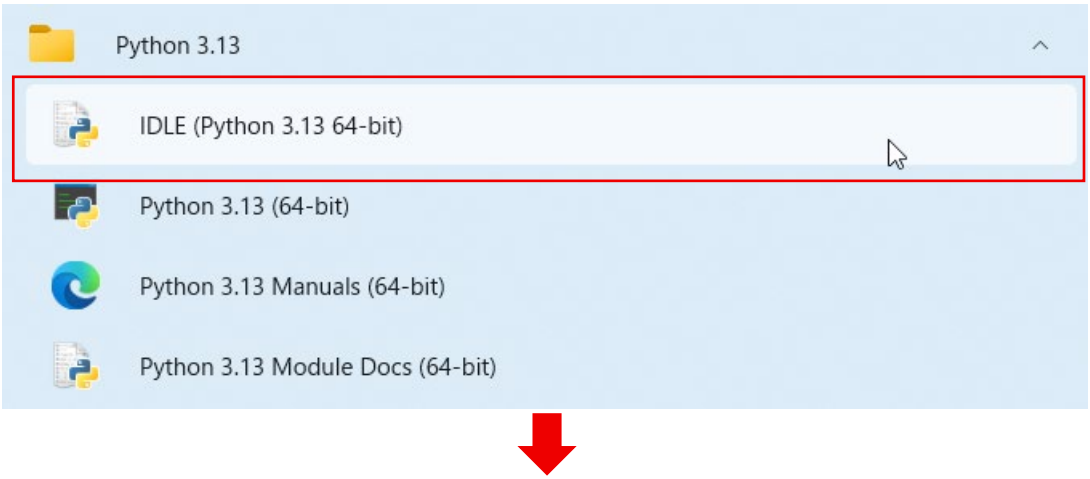
プログラムの作成にあたっては，Python をダウンロードしてインストールするか，または Google Colaboratory にアクセスすること。なお，ここでは 2025 年時点で最新の Python3.13.7 のバージョンで説明する。

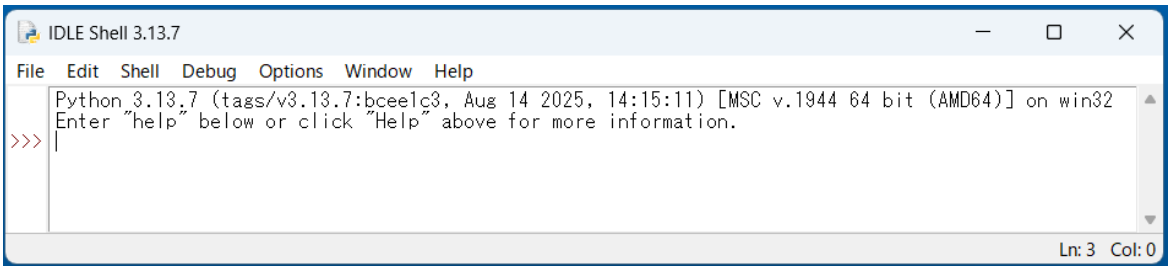
２．IDLE の基本操作

Python に標準で付属している IDLE（アイドル）の基本的な使い方を説明する。IDLE は，プログラムの実行結果を表示する「シェルウィンドウ」とプログラムを編集する「エディタウィンドウ」の２種類で構成されている。IDLE の基本的な操作は次のとおりである。

（１）IDLE の起動

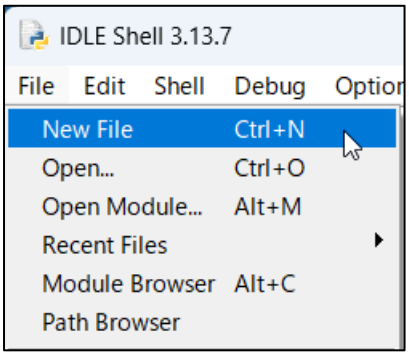
[スタート]－[Python 3.13]－[IDLE(Python 3.13 64-bit)]をクリックする。タイトルバーに「IDLE Shell 3.13.7」とあるシェルウィンドウが表示される。





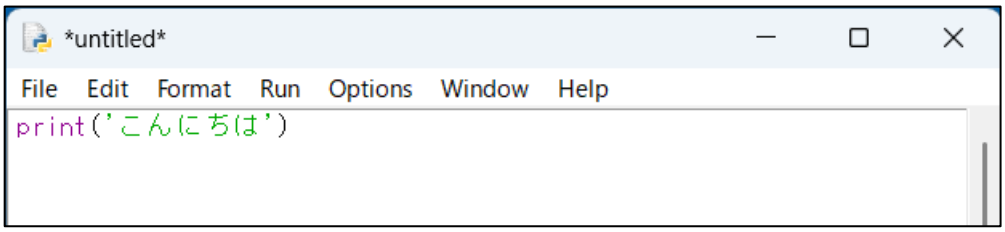
（２）エディタウィンドウの表示

シェルウィンドウのメニューバーにある[File]－[New File]をクリックする。プログラムを記述するエディタウィンドウが表示される。



（3）プログラムの記述

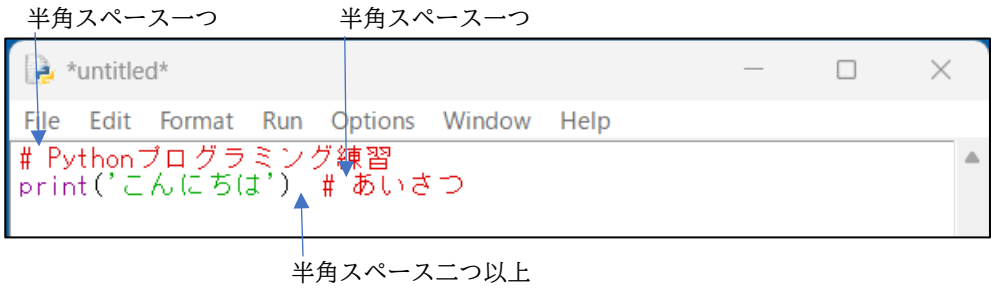
プログラムは基本的にエディタウィンドウに記述して保存する。



（4）コメント

コメントとは，プログラムの中に記述する説明やメモのことである。コメントを入力しておくと，後でプログラムを編集するとき，どのような処理をしていたのか思い出しやすい。また，ほかの人がプログラムを見たときも処理内容がわかりやすくなる。

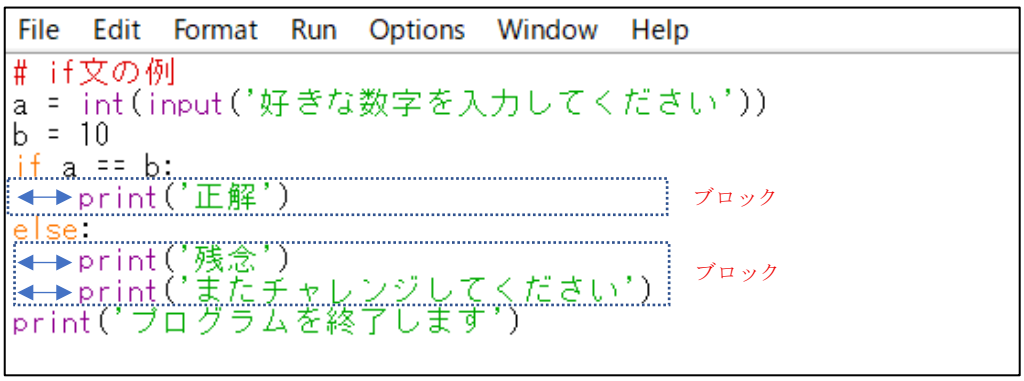
Python でコメントを入れるときには，#を使用する。#から行末までの記述は，実行時に無視される。#を行頭に書いた場合は，その行すべてがコメントと見なされるため，実行時に無視される。



コメントを記述するときは，#の後に半角スペースを一つ入れる。また，文とコメントの間は，少なくとも半角スペースを二つ入れる。なお，半角スペースを入れなくても動作するが，半角スペースを入れることは，Python のプログラムを記述するうえでの推奨ルールとなっている。

（5）インデント

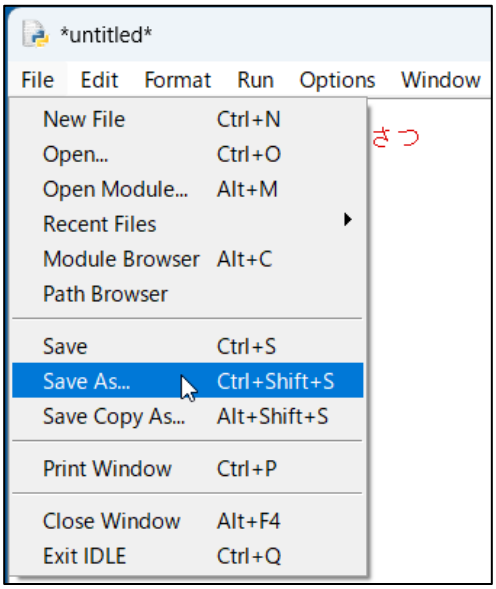
行頭の字下げをインデントといい，Python では，後述の関数の定義や if 文を記述するときなどに，インデントを用いて処理のまとまりを表す。一般的には，行頭から半角 4 文字分の字下げを行う。字下げはスペースキーを 4 回押すか，Tab キーを使う。Tab キーは，IDLE の初期設定では，1 度押すだけで半角 4 文字分の字下げとなる。インデントした範囲のことをブロックと呼ぶ。



※◀▶ はインデント（実際のプログラムには矢印は表示されない）

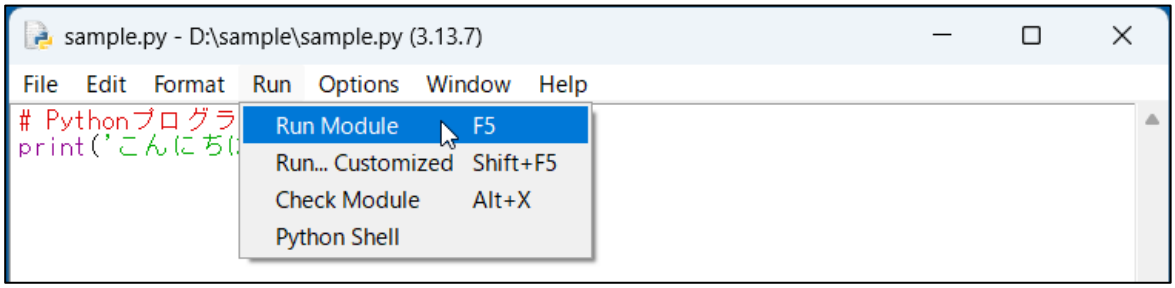
（6）プログラムの保存

[File]－[Save As…]をクリックし，名前を付けてファイルを保存する。

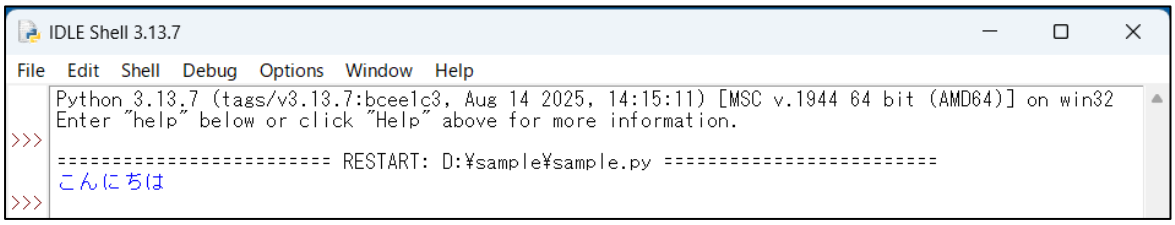


（7）プログラムの実行

①エディタウィンドウのメニューバーにある[Run] －[Run Module]をクリックするか，F5 キーを押す。

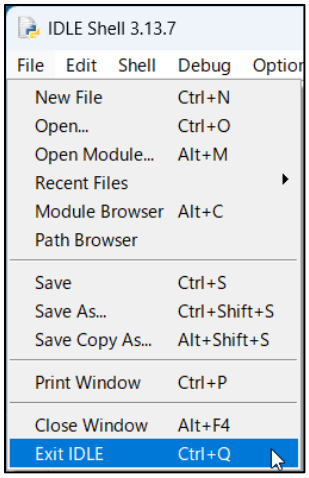


②シェルウィンドウに実行結果が表示される。



（8）IDLE の終了

シェルウィンドウまたはエディタウィンドウのメニューバーから[File]－[Exit IDLE]をクリックする。シェルウィンドウとエディタウィンドウの双方ともウィンドウが閉じ，IDLE が終了する。



３．変数

Python は，変数の型や名前などを事前に記述(変数の宣言という)する必要がなく，使用する箇所で変数名を記述するだけで変数を利用することができる。Python では，変数名として半角英小文字を使用することが推奨されている。

（１）変数名のルール

変数に使用できる名前(変数名という)には，次のルールがある。

①変数名は半角の英小文字で始める。
　※数字を使うこともできるが，「数字のみ」や「先頭が数字」の変数名は使用できない。
　※英大文字と英小文字は別の文字として扱われる。
　※アンダースコア(_)で始めることは可能だが推奨されていない。
　※全角文字を使うことはできるが推奨されていない。
②予約語は変数名として使用できない

予約語とは，Python で使用目的が決まっているため，変数名として使用できない語句のことである。

■Python の予約語一覧

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

このほか，Python に標準で用意されている組み込み関数と同じ名前の変数を使用することもエラー発生の原因となるため，推奨されていない。なお，組み込み関数については後述する。

■組み込み関数一覧

```
abs, aiter, all, anext, any, ascii, bin, bool, breakpoint, bytearray,
bytes, callable, chr, classmethod, compile, complex, delattr, dict, dir,
divmod, enumerate, eval, exec, filter, float, format, frozenset, getattr,
globals, hasattr, hash, help, hex, id, input, int, isinstance,
issubclass, iter, len, list, locals, map, max, memoryview, min, next,
object, oct, open, ord, pow, print, property, range, repr, reversed,
round, set, setattr, slice, sorted, staticmethod, str, sum, super, tuple,
vars, zip, __import__
```

（２）変数への代入

変数に値を入れることを代入といい，＝を使って記述する。

値の種類	例	記述
数値	変数 a に「10」を代入	`a = 10`
文字列	変数 b に「こんにちは」を代入	`b = 'こんにちは'`

※文字列は ' （シングルクォーテーション)または " （ダブルクォーテーション)で囲む。

（3）データ型

プログラミング言語では，一般的に変数を使用する前に，その変数に代入する値の型(データ型)を宣言するが，Python の場合は，最初に変数に入れた値でその型が決まるため，事前に宣言しなくても変数を利用することができる。しかし，プログラムを作成するうえでデータ型を指定しなければいけない場合や，データ型をプログラムの途中で変更しなければいけない場合もあるため，データ型を意識しながらプログラミングを行うとよい。おもなデータ型を紹介する。

データ型（一部）

データ型	内容	使用例
int	整数を表す型。	a = 10
float	実数を表す型。	a = 3.14
str	文字列を表す型。	a = 'こんにちは'
bool	真偽値（条件が成立しているかどうか）を表す型。 True または False のどちらかを表す。	a = True
list	複数の値を入れることができる型。 値の追加や削除ができる。	a = [10, 20, 30, 40, 50]
tuple	複数の値を入れることができる型。 値の追加や削除ができない。	a = ('東京', '大阪', '福岡')
dict	キーと値の二つで一つの組になる型。 複数の組から構成される「辞書」を表現する型。 キーと値の組は追加や削除ができる。	a = {'東京':10, '大阪':20}
set	集合を扱うための型。 値の追加や削除ができる。	a = set([10, 20, 30])

４．関数

関数とは，あらかじめ定義されている処理を行い，その結果を返す機能である。Python の関数には，標準で用意されている組み込み関数とユーザが自ら作成するユーザ定義関数がある。

一般的に，関数を利用するときは，引数と呼ぶ値を関数に渡す。関数は受け取った引数に基づいて処理を行う。なお，関数は処理した結果を値として返す場合がある。この値を戻り値という。

【書式】

関数名(引数１，引数２，引数３，…)

（1）組み込み関数

組み込み関数とは，すぐに使用することが可能な Python に標準で用意されている関数のことである。よく利用される組み込み関数を紹介する。

出力を操作する関数

print 関数
（書式）print(出力する内容)
（解説）文字列や計算した結果などを，画面またはファイルに出力する。

【使用例】

例	コード	実行結果	解説
1	`print(10)`	`10`	カッコ内の数値が出力される。
2	`print('こんにちは')`	こんにちは	文字列を出力するときは'または"で囲む。
3	`print(100, '円')`	`100 円`	数値と文字列を並べて出力させるときは， ,(コンマ)を使って出力させる値を指定する。 この場合，数値の後ろに半角スペースが表示される。
	`print('100' + '円')`	`100円`	数値を'または"で囲み，＋を使って文字列として連結する。

4	`a = 10`
	`b = 20`
	`print(a)`
	`print(a + b)`
		
		`10`
		`30`
			
			変数 a に代入された値を出力。
			変数 a と変数 b の足し算をした結果を出力。

※演算子+は数値の場合は「足し算」，文字列の場合は「連結」となる。

文字列に変換する関数

str 関数
（書式）`str(`文字列に変換する数値または変数`)`
（解説）引数に指定した数値または変数を文字列に変換する。

【使用例】

コード	実行結果
`price = 100` `print('牛乳は' + str(price) + '円です。')`	牛乳は `100` 円です。

（解説）

変数 `price` に代入されている値は数値である。このままでは文字列とつなげることができない。そこで str 関数を使って変数 `price` に格納されている値を文字列に変換し，前後の文字列と連結する。

ユーザと対話する関数

input 関数
（書式）`input('メッセージ')`
（解説）ユーザに入力を促すメッセージを表示する。 ユーザが入力した文字列を返す。

【使用例】

コード	実行結果と入力操作
price = input('牛乳の価格を入力してください。') print('牛乳は' + price + '円です。')	牛乳の価格を入力してください。 「100 と入力し，Enter キーを押す」 牛乳は 100 円です。

（解説）

input 関数に画面に表示したいメッセージを引数として渡す。ユーザが入力したデータは戻り値として返り，変数 price に代入される。なお，input 関数を使って入力した値のデータ型は文字列となる。そのため，変数 price はそのまま前後の文字列と連結することができる。

数値を操作する関数

int 関数
（書式）int(データ)
（解説）引数に指定した数値または文字列を整数に変換する。 実数が指定された場合は，小数点以下を切り捨てた整数になる。

float 関数
（書式）float(データ)
（解説）引数に指定した数値または文字列を実数に変換する。

【使用例】

コード	実行結果と入力操作
price =input('牛乳の価格を入力してください。') zeikomi = float(price) * 1.1 zeikomi = int(zeikomi) print('牛乳は税込' + str(zeikomi) + '円です。')	牛乳の価格を入力してください。 「100 と入力し，Enter キーを押す」 牛乳は税込 110 円です。

（解説）

input 関数を使用すると，ユーザが入力した値は文字列として返る。そのため計算に使用するときは，整数や実数などのデータ型へ変換する必要がある。そこで，float(price)で文字列を実数に変換し，1.1 倍して税込み価格を求め，その結果を変数 zeikomi に代入する。変数 zeikomi は小数点以下の値も含まれているため，str 関数で文字列に変換した場合は，そのまま小数点以下の値も表示される。

そこで，int 関数を使って変数 zeikomi にある小数点以下の数値を切り捨てて整数にする。前後の文字列と連結させて画面に出力するため，str 関数を使って変数 zeikomi にある数値を再び文字列に変換する。

（2）ユーザ定義関数

同じような処理をプログラムの中で何度も行うときは，関数にすると便利である。ユーザが自ら作成した関数をユーザ定義関数という。関数は，次のように定義する。

関数の定義	
（書式） def 関数名(引数 1，引数 2，引数 3，…): 　　関数で実行する処理 　　return(戻り値 1，戻り値 2，戻り値 3，…)	（解説） 関数で処理したい値をコンマで区切る。 半角 4 文字分をあけてから記述する。 処理後に返す値を設定する。

関数の利用(上記で定義した関数の戻り値が入る)
（書式） 変数 = 関数名(引数 1，引数 2，引数 3，…)

【使用例】

コード	実行結果

# 関数の定義 	
def plus(a, b):
　　c = a + b
　　return(c)
# 関数の利用 	
d = plus(10, 20)
e = plus(1.5, 4.6)
print(d)
print(e)
	
	30
	6.1

（解説）

関数の定義

「# 関数の定義」以降の 3 行で，plus()関数というユーザ定義関数の定義を行っている。この例では，plus()関数は，引数として a と b の値を受け取り，a と b を足し算した結果を返すよう設定している。

関数の定義を行うとき，関数で実行する処理以降は，半角 4 文字分の字下げ(インデント)をしてから記述する。なお，一つのプログラムの中で複数の関数を定義することができるが，関数の定義は，必ず呼び出す箇所より先に記述しなければならない。

関数の利用

「# 関数の利用」以降は，定義した plus()関数を利用した例である。ここでは，引数として渡した値が処理されて戻ってきた値を，変数 d と変数 e にそれぞれ代入し，print 関数を利用して画面に出力している。

5．エスケープシーケンス

例えば，「It's mine」という文字列を表示する場合，print('It's mine')と記述すると，エラーが発生する。これは'(シングルクォーテーション)が出力したい文字列の中にもあることが原因で発生したエラーである。このようなときには，エスケープシーケンスという表記方法を利用する。

エスケープシーケンスを表す記号として，一般に\(バックスラッシュ)を使用するが，OS が Windows の場合は，¥(円記号)を使用する。

【使用例：Windows の場合】

例	コード	実行結果	解説
1	print('It¥'s mine')	It's mine	エスケープシーケンスを使用せずに"It's mine"と，"(ダブルコーテーション)を使って文字列全体を囲んで記述する方法もある。
2	print('¥¥10,000')	¥10,000	¥はエスケープシーケンスを表す記号である。そのため¥10,000 と出力したい場合は，左のように記述する。このほか print(r'¥10,000')と記述する方法もある。r は raw(ロー)文字列と呼ばれ，¥をエスケープシーケンスの記号として扱わず，一つの文字として扱う。 なお，print('¥10,000')と記述した場合，実行結果は「・,000」となる。
3	print('1¥n2¥n3')	1 2 3	¥n は改行の指示を表す。 左のように記述すると，改行されて出力される。

【よく利用されるエスケープシーケンスと意味】

記号	Windows の場合	意味
\'	¥'	'（シングルクォーテーション）
\"	¥"	"（ダブルクォーテーション）
\\	¥¥	\（バックススラッシュ）　Windows は¥(円記号)
\n	¥n	改行
\t	¥t	タブ

エスケープシーケンスを使うと，かえってプログラムが読みにくい場合は，'(シングルクォーテーション)または"(ダブルクォーテーション)を前後に三つずつ入力して囲むとよい。これを三重引用符(三重クオート)という。この場合，改行も含めて入力したとおりに出力される。

コード	実行結果
print('''桃太郎さん　桃太郎さん お腰につけた　きびだんご 一つわたしに　くださいな''')	桃太郎さん　桃太郎さん お腰につけた　きびだんご 一つわたしに　くださいな

6．演算子

計算式で使用する記号のことを演算子という。使用するときは，日本語入力をオフにし，必ず半角で入力する。なお，演算子の前後に半角スペースを入れても問題なく動作するので，入力したコードを読みやすくするため，一般的に演算子の前後に半角スペースを入れて記述することが多い。

演算子には次のようなものがある。

【代入演算子】

代入演算子とは，変数に値を代入する演算子のことで＝(イコール)を使用する。これは左辺と右辺が等しいという意味ではなく，左辺に右辺の値を格納するという意味である。

演算子	内容	利用例	解説
=	右辺の値を左辺に代入	`total = 0`	変数 `total` に値 `0` を代入。

【算術演算子】

演算子	内容	使用例	結果	解説
+	加算(足し算)	`print(1 + 2)` `print(1 + 2.0)`	3 3.0	整数と実数を足した結果や，整数と実数を引いた結果は，実数で表示される。
-	減算(引き算)	`print(5 - 3)` `print(5 - 3.0)`	2 2.0	
*	乗算(掛け算)	`print(2 * 3)`	6	
/	除算(割り算)	`print(10 / 2)`	5.0	結果は実数で表示される。
//	整数除算(整数の割り算)	`print(10 // 2)` `print(10 // 3)`	5 3	小数点以下を切り捨てた割り算となる。
%	剰余	`print(10 % 3)`	1	割り算の余りを返す。
**	べき乗	`print(2 ** 3)`	8	

【累算代入演算子(複合代入演算子)】

累算代入演算子(複合代入演算子)を使用すると，通常の式を省略して記述することができる。

演算子	内容	使用例	通常の式	解説
+=	加算代入	`x += 1`	`x = x + 1`	x に x＋1 をした値を代入。
-=	減算代入	`x -= 1`	`x = x - 1`	x に x－1 をした値を代入。
*=	乗算代入	`x *= 5`	`x = x * 5`	x に x×5 をした値を代入。
/=	除算代入	`x /= 5`	`x = x / 5`	x に x÷5 をした値を代入。
//=	整数除算代入	`x //= 3`	`x = x // 3`	x に x÷3 をした値のうち小数点以下を切り捨てて代入。
%=	剰余代入	`x %= 3`	`x = x % 3`	x に x÷3 をして生じた余りを代入。
**=	べき乗代入	`x **= 5`	`x = x ** 5`	x に x の 5 乗をした値を代入。

【比較演算子】

左辺と右辺の二つの値を比較して判定する演算子である。条件によって処理を変えるときに使用する。条件が成立した場合は True，成立しない場合は False を返す。

演算子	内容	使用例	解説
==	左辺と右辺が等しい	a == b	a と b の値が等しい。
>	左辺が右辺より大きい	a > b	a の値は b の値より大きい。
>=	左辺が右辺以上	a >= b	a の値が b の値以上。
<	左辺が右辺より小さい(未満)	a < b	a の値が b の値より小さい。
<=	左辺が右辺以下	a <= b	a の値が b の値以下。
!=	左辺と右辺が等しくない	a != b	a の値と b の値は等しくない。
is	左辺と右辺が同じである	a is b	a と b は同じオブジェクトである。
is not	左辺と右辺は同じではない	a is not b	a と b は同じオブジェクトではない。
in	左辺の要素が右辺にある	a in b	a という要素が b に存在する。
not in	左辺の要素は右辺にない	a not in b	a という要素が b に存在しない。

（解説）

==演算子では，オブジェクトの値が同じかどうか判定するのに対し，is 演算子はオブジェクトが同じかどうか判定する。

【論理演算子(ブール演算子)】

複数の条件式を作成するときに使用する。

演算子	内容	使用例	解説
and	すべての条件を満たすとき True そうでなければ False を返す	a == 1 and b <= 5	a の値が 1 で，かつ b の値が 5 以下
or	いずれかの条件を満たすとき True そうでなければ False を返す	a == 1 or b <= 5	a の値が 1，または b の値が 5 以下
not	条件を「ではない」と否定 否定した条件どおりであれば True そうでなければ False を返す	not b <= 5	b の値が 5 以下ではない

【文字列に使用する演算子】

演算子	内容	使用例	実行例
+	文字列をつなげる(連結)	name = '桃太郎' print(name + 'さん')	桃太郎さん
*	指定した回数だけ文字列を繰り返して表示する	word = 'ぽっ' print(word * 3 + '鳩ぽっぽ')	ぽっぽっぽっ鳩ぽっぽ

7．基本構文

（1）条件分岐(分岐処理(選択処理))

if
【書式】 if 条件: 　　条件が成立したときに実行する処理

【使用例】

コード	実行結果
if a < b: 　　print('b が大きいです')	(条件が成立したときは下記を表示) b が大きいです

(解説)

if 文を用いると条件に応じた処理が行える。条件は式や関数を用いて記述する。条件が成立したときに実行する処理は必ずインデントしてから記述する。インデントをしないで行頭から記述するとエラーになる。

if～else

【書式】
if 条件:
　　条件が成立したときに実行する処理
else:
　　条件が成立しないときに実行する処理

【使用例①】

コード	実行結果

if a < b:
　　print('b が大きいです')
else:
　　print('b は a と等しいか小さいです')
	(条件が成立したときは下記を表示)
	b が大きいです
	(条件が成立しないときは下記を表示)
	b は a と等しいか小さいです

【使用例②】

コード	実行結果

if a == b:
　　pass
else:
　　print('a と b は等しくありません')
	(条件が成立したときは何も実行しない)
	
	(条件が成立しないときは下記を表示)
	a と b は等しくありません

(解説)

else:を記述すると，条件が成立しないときの処理を記述することができる。else:は if と行頭の位置を揃えて記述し，条件が成立していないときに実行する処理は，必ずインデントしてから記述する。

条件の成立時，あるいは条件の不成立時に，何も処理をしない場合は pass と記述する。なお，pass の記述は省略することができる。

if～elif～else

【書式】
if 条件 1:
　　条件 1 が成立したときに実行する処理
elif 条件 2:
　　条件 2 が成立したときに実行する処理
elif 条件 3:
　　条件 3 が成立したときに実行する処理
 	
（以下，条件の数だけ同様に記述）
 	
else:
　　すべての条件が成立しないときの処理

【使用例】

コード	実行結果

if a < b:
　　print('b が大きいです')
elif a == b:
　　print('a と b は等しいです')
else:
　　print('b が小さいです')
	（条件 1 が成立したときは下記を表示）
	b が大きいです
	（条件 2 が成立したときは下記を表示）
	a と b は等しいです
	（すべての条件が不成立の場合は下記を表示）
	b が小さいです

（解説）

elif は else と if を合わせたものを短縮した形で，条件によって処理が三つ以上に分岐する場合に使用する。elif は if と行頭の位置を揃えて記述し，条件が成立したときに実行する処理は，必ずインデントしてから記述する。elif は必要な数だけ複数記述することができる。

すべての条件が成立しないときの処理を記述する必要がないときは，else:を省略することができる。

（２）for 文による反復処理(繰り返し処理)

for
【書式】 for 変数 in オブジェクト: 　　繰り返し実行する処理

【使用例】

コード	実行結果
for i in range(3): 　　print('ぽっ') print('鳩ぽっぽ')	ぽっ ぽっ ぽっ 鳩ぽっぽ

（解説）

for 文は指定した回数だけ処理を繰り返すときに使用する。変数は繰り返す数をカウントするのに使用する。オブジェクトで繰り返す回数を指定する。繰り返す回数は range 関数を用いて指定することが多い。

繰り返し実行する処理は，必ずインデントしてから記述する。Python はインデントされた部分が for 文で実行されるブロックと認識している。

range 関数
（書式）range(開始する値,終了する値,増分)
（解説）連続した数値のオブジェクトを作成する。 ①開始する値は省略可能である。 ②開始する値を指定しない場合，0 から開始する。 ③終了する値に指定した値未満(終了する値の一つ前)まで繰り返す。 ④増分は増やす値が 1 ならば省略可能である。 ⑤増分には負の値を指定することができる。

【使用例】

コード	実行結果
`for i in range(1, 10, 2):` `  print(i)`	1 3 5 7 9

（3）while 文による反復処理(繰り返し処理)

while
【書式】 while 条件: 　条件が成立している間は繰り返して実行する処理

【使用例①】

コード	実行結果

`a = 1`
`while a <= 3:`
`  print('ぽっ')`
`  a = a + 1`
`print('鳩ぽっぽ')`
	ぽっ
	ぽっ
	ぽっ
	鳩ぽっぽ

（解説）

while 文を用いると，条件が満たされている限り，インデントして記述したブロックの処理を繰り返し実行する。反復処理を行う部分では，条件判定に使用する変数の値などを変更することで，次の条件判定のときに繰り返し処理を継続するか，while 文を終了するかが決まる。

【使用例②】

コード	実行結果

`a = 0`
`while True:`
`  print('ぽっ')`
`  a = a + 1`
`  if a >= 3:`
`  break`
`print('鳩ぽっぽ')`
	ぽっ
	ぽっ
	ぽっ
	鳩ぽっぽ

（解説）

while 文の条件には，True や False を指定することもできる。True を指定した場合は，永遠に条件が成立することになる。この場合，while 文のループから抜けるには break を使用する。使用例②の場合は，if 文で条件を記述し，条件が成立した場合は break でループを抜けている。

8．リスト

int 型や str 型の変数は，一つの変数名に対して一つの値しか保存できない。しかし，リストを用意すれば複数の値を一つのリスト名で保存しておくことができる。一般的に，リストとは，用意した箱の中に仕切りがあり，その仕切りごとに値が保存されているものと考えるとよい。それぞれの仕切りの中に代入された値のことを要素という。要素は 0 から順番に番号が割り当てられた各仕切りの中に入る。この番号を添字(インデックス)という。添字を指定することで各要素を扱うことができる。リストを作成するときは，全体を[ ]で囲み，要素はコンマで区切る。リストから要素を参照するときは，リスト名の後に添字を[ ]で囲って指定する。

【使用例①】

コード	実行結果
a = [40, 60, 50, 70] print(a[2])	（リスト a に四つの要素を代入） 50

（解説）

リスト a に四つの要素を代入後，リスト a の添字 2 を指定し，そこに格納されている値を表示している。添字は 0 から始まる。格納された値の先頭から 2 番目ではないことに注意する。

【使用例②】

コード	実行結果
name = ['桃', '金', '浦島'] for a in name: 　print(a + '太郎')	（リスト name に三つの要素を代入） （name にある要素を一つずつ変数 a に代入） 桃太郎 金太郎 浦島太郎

（解説）

for 文を使ってリストの中にある要素を一つずつ取り出し，太郎の文字と連結して出力している。

【使用例③】

コード	実行結果
data = [80, 70, 55, 65, 75] goukei = 0 heikin = 0 for a in data: 　goukei = goukei + a heikin = goukei / len(data) print('平均は' + str(heikin) + '点')	（リスト data にテストの得点を代入） （変数 goukei を初期化） （変数 heikin を初期化） （リスト data の値を一つずつ変数 a に代入） （変数 goukei に各要素の値を加算） （合計した値を要素数で割り，変数 heikin に代入） 平均は 69.0 点

（解説）

リスト data にテストの得点を格納する。for 文を使って変数 goukei にリストから取り出した得点を一つずつ加算する。リストにあるすべての要素の加算が終わるとループを抜け，平均点を算出する。平均点を算出するときに使用した len は関数で，リスト内にある要素数を数えている。

len 関数		
（書式）len(引数)		
（解説）引数に指定したオブジェクトの長さや要素の数を取得する。引数には文字列やリスト，タプル，辞書などのオブジェクトを指定することができる。		
コード	実行結果	解説
`print(len('Tokyo'))`	5	文字列 Tokyo は半角 5 文字である。
`print(len(['Tokyo', 'Nagoya', 'Kyoto']))`	3	リストには三つの要素がある。

【使用例④ 】

コード	実行結果

`data = [80, 70, 55, 65, 75]`
`tuika = [60, 90]`
`data = data + tuika`
`print(data)`
	（リスト data にテストの得点を代入）
	（リスト tuika に 60 と 90 を代入）
	（リスト data の要素の後にリスト tuika の要素を追加）
	`[80, 70, 55, 65, 75, 60, 90]`

（解説）

＋を使ってリストどうしを足すことができる。リストに要素を追加するときに用いる。

【使用例⑤ 】

コード	実行結果

`data = [80, 70, 55, 65, 75]`
`data.append(60)`
`print(data)`
	（リスト data にテストの得点を代入）
	（リスト data の末尾に 60 を追加）
	`[80, 70, 55, 65, 75, 60]`

（解説）

append メソッドを使い，リストの末尾に要素を追加することができる。

append メソッド
（書式）操作対象.append(引数)
（解説）引数に追加する要素を指定することで，操作対象(この場合はリスト)の末尾に要素を追加ができる。

【使用例⑥ 】

コード	実行結果

`data = [80, 70, 55, 65, 75]`
`del data[1]`
`print(data)`
	（リスト data にテストの得点を代入）
	（リスト data にある添字 1 の要素を削除）
	`[80, 55, 65, 75]`

（解説）

del 文を使ってリストにある要素を削除することができる。

del
（書式）del 削除する変数やオブジェクト１，削除する変数やオブジェクト２,…
（解説）del 文は変数やオブジェクトなどを削除するときに使用する。リストやディクショナリのときは，削除したい要素を指定する。

【使用例⑦ 】

コード	実行結果
data = [80, 70, 55, 65, 75] a = sorted(data) print(a)	（リスト data にテストの得点を代入） （リスト data のデータを昇順に並びかえて変数 a に代入） [55, 65, 70, 75, 80]

（解説）

sorted 関数を使ってリストにある要素を昇順(小さい順)に並び替えている。

sorted 関数	
（書式）sorted(引数)	
（解説）引数には並び替えるリストを指定する。二つ目の引数に reverse = True と指定するとリストを降順(大きい順)に並び替える。	
コード	実行結果
data = [80, 70, 55, 65, 75] a = sorted(data, reverse = True) print(a)	（リスト data にテストの得点を代入） （リスト data のデータを降順に並び替えて変数 a に代入） [80, 75, 70, 65, 55]

9．タプル

タプルは，リストと同じように複数の値をもつことができる。しかし，リストと違って一度定義すると，後から値を変更したり，要素の追加や削除をしたりすることができない。要素が変わることがなく，誤って要素を書き換えることがないようにしたいとき，タプルを使用するとよい。

【使用例】

コード	実行結果
a = (40, 60, 50, 70) print(a[2])	（タプル a に四つの要素を代入） 50

（解説）

タプル a に値を代入するときは，()で要素を囲み，各要素はコンマで区切る。なお，定義するときの()は省略することができるため，a = 40, 60, 50, 70 と記述してもよい。2 行目の print(a[2])は添字 2 に格納されている値を表示している。添字は，リストと同様に 0 から始まる。

１０．スライス

スライスとは，文字列，リスト，タプルなどにあるデータの一部を，添字を用いて取り出す操作である。

【使用例①】

コード	実行結果
a = 'お腰につけたきびだんご一つ私にくださいな' print(a[11:13])	（変数 a に文字列を代入） 一つ

（解説）

変数 a に文字列を代入後，変数 a の添字 11 から 13 までを指定し，そこに格納されている値を出力している。添字は 0 から始まり，文字列の場合は 1 文字ずつ添字が割り当てられるため，添字 11 にある要素は「一」である。出力する範囲の終わりに指定する添字は指定した数値の前までになる。つまり，ここでは 13 を指定したので添字 12 にある「つ」までを出力する。

スライス
（書式）変数名[start:stop:step]
（解説） 添字の区切りは:(コロン)を使う。 start　開始する添字 stop　終了する添字＋１　（指定した添字の一つ前までが範囲になるため） step　何個ごとに抽出するかを指定（省略すると一つずつ）

【使用例②】

コード	実行結果
a = [0, 1, 2, 3, 4, 5, 6, 7, 8]	（リスト a に九つの要素を代入）
print(a[1:4])	[1, 2, 3]
print(a[1:8:2])	[1, 3, 5, 7]
print(a[:7])	[0, 1, 2, 3, 4, 5, 6]
print(a[7:])	[7, 8]
print(a[-1:-5:-1])	[8, 7, 6, 5]

（解説）

リストにあるデータの一部を取り出す例である。リスト a に値を代入後，添字を用いて，そこに格納されている値を出力している。

print(a[1:4])は，添字 1 から添字 3 までの三つの要素を出力する。

print(a[1:8:2])は，添字 1 から添字 7 までの範囲にある要素を一つおきに取得して出力する。

print(a[:7])は，開始する添字を省略すると，最初から添字 6 までを出力する。

print(a[7:])は，添字 7 以降を指定し，出力する。

print(a[-1:-5:-1])は，開始する添字を-1 と指定している。添字は後ろから指定することもでき，start を-1 と指定すると末尾の要素を取得する。また，step を-1 と指定することで，リストを末尾から順番に出力することができる。

リストの要素	0	1	2	3	4	5	6	7	8
インデックス	-9	-8	-7	-6	-5	-4	-3	-2	-1

【使用例③】

コード	実行結果
a = (0, 1, 2, 3, 4, 5, 6, 7, 8) b = a[1:4] print(b)	（タプル a に九つの要素を代入） （タプル b にタプル a の添字 1 から添字 3 までを代入） (1, 2, 3)

（解説）

スライス機能を使ってタプルから指定した範囲の要素が含まれる新しいタプルを作成している。

タプルは要素の変更ができないため，必要なときには，このように新しくタプルを作成する。

１１．ディクショナリ(辞書)

ディクショナリは辞書ともいい，リストが添字を使って要素を取り出すのに対し，キー(任意の文字列)を使ってデータを取り出す。

ディクショナリは，{ } の間に，キーと対応する値の組み合わせを キー:値 の形式で記述し，複数の要素を ,(コンマ)で区切って定義する。

【使用例①】

コード	実行結果
a = {'桃太郎':40, '金太郎':60} print(a['金太郎'])	(ディクショナリ a に，キーを名前，値を得点として代入) 60

(解説)

一行目でディクショナリ a に生徒の名前をキーに値を代入した後，二行目でキーにした生徒の名前を指定して値を参照し，出力している。

【使用例②】

コード	実行結果
a = {'桃太郎':40, '金太郎':60,} key = a.keys() value = a.values() all = a.items() print(key) print(value) print(all)	(ディクショナリ a に，キーを名前，値を得点として代入) (keys 関数を使い，すべてのキーをリスト key に代入) (values 関数を使い，すべての値をリスト value に代入) (items 関数を使い，すべてのキーと値をディクショナリ all に代入) dict_keys(['桃太郎', '金太郎']) dict_values([40, 60]) dict_items([('桃太郎', 40), ('金太郎', 60)])

(解説)

keys 関数を使用すると，ディクショナリからキーのみ取得することができる。values 関数を使用すると，ディクショナリから値のみを取得することができる。items 関数を使用すると，キーと値を辞書から同時に取得することができる。

keys 関数	(書式) keys()	(解説) ディクショナリのキーを返す。
values 関数	(書式) values()	(解説) ディクショナリの値を返す。
items 関数	(書式) items()	(解説) ディクショナリのキーと値を返す。

【使用例③】

コード	実行結果
a = {'桃太郎':40, '金太郎':60,} a['浦島太郎'] = 50 print(a)	(ディクショナリ a に名前と得点を代入) ディクショナリ a に，キーが浦島太郎，値が 50 の組を追加 {'桃太郎':40, '金太郎':60, '浦島太郎':50}

(解説)

ディクショナリは後からキーと値を追加することができる。なお，キーと値を削除したいときには del 文を使用する。

１２．集合

Python では，何らかの値が集まったものを集合と呼ぶ。集合は要素に順番がなく，また，一つの集合には同じ要素が含まれないという性質をもつ。集合は，ある集合に特定の要素が含まれているか調べたり，ある集合と別の集合を比較したりする目的で使用することが多い。例えば，リストやディクショナリから重複する値を取り除くために集合を使うことがある。そのため，集合の和，差，積などを計算するメソッドや演算子が Python には用意されている。集合を定義するときは，ディクショナリと同じように{}を使う。

【使用例】

コード	実行結果
`a = {1, 2, 2, 2, 3, 3, 4, 4}`	（集合 a にデータを集合として値を代入）
`b = {1, 2, 2, 2, 5, 5, 6, 6}`	（集合 b にデータを集合として値を代入）
`print(a)`	`{1, 2, 3, 4}`
`print(b)`	`{1, 2, 5, 6}`
`print(a | b)`	`{1, 2, 3, 4, 5, 6}`

（解説）

集合 a および集合 b にデータを代入している。そのため三行目および四行目では，重複する値は無視されて一意な値のみが出力される。5 行目では，演算子の|(パイプ)を用いて集合 a と集合 b の和集合を出力している。

集合演算	説明	演算子
和集合	二つ以上の集合に含まれているすべての要素を集めた集合。	|
差集合	ある集合から，別の集合に含まれている要素を除いた集合。	-
積集合	二つの集合の両方に含まれる要素の集合。	&
対称差集合	二つの集合のどちらか片方だけに含まれる要素の集合。	^

１３．モジュール

モジュールとは，関連性のあるプログラムが一つのファイルとしてまとめられたもので，Python では，さまざまな機能を提供するモジュールが標準ライブラリとして用意されている。標準ライブラリを使用するときは，import 文を使って自分のプログラムに導入する。

モジュールをインポート後，そのモジュールが提供している機能を利用するときは，モジュール名の後に.(ドット)を付け，その後に関数などを指定する。

【使用例①】

コード	実行結果
`import random` `print(random.random())`	（random モジュールを導入） （0 以上 1 未満の任意の値が出力される）

（解説）

random は乱数を生成するモジュールである。一行目で import 文を使って random モジュールを導入し，2 行目で random モジュールに用意されている random 関数を使って 0 以上 1 未満の小数の乱数を一つ取り出して出力する。

import
【書式】import モジュール名

random 関数
（書式）random.random()
（解説）0.0 から 1.0 までの範囲の float 型の値を返す関数である。random 関数を利用するには「モジュール名.関数名」と記述をする必要があるため，random.random()となる。

【使用例②】

コード	実行結果
import random print(random.randint(1,10))	（random モジュールを導入） （1 から 10 までの任意の整数が出力される）

（解説）

整数の乱数を生成したいときは random モジュールに用意されている randint 関数を使用する。ここでは，1 から 10 までの範囲で整数の乱数を一つ取り出して出力する。

なお，指定した範囲内で小数の乱数を生成したいときは，random モジュールに用意されている uniform 関数を使用する。randint の箇所を uniform と記述すればよい。

randint 関数
（書式）random.randint(a, b)
（解説）a から b までの指定した範囲のランダムな整数を返す関数である。randint 関数を利用するには「モジュール名.関数名」と記述をする必要があるため，random.randint(a, b)となる。

uniform 関数
（書式）random.uniform (a, b)
（解説）a から b までの指定した範囲のランダムな小数を返す関数である。uniform 関数を利用するには「モジュール名.関数名」と記述をする必要があるため，random.uniform(a, b)となる。

なお，標準ライブラリ以外に別に配布されているモジュールのことを，外部モジュールや外部ライブラリなどといい，これらを利用するときも同じように import 文を用いて導入する。

１４．デバッグ

デバッグ(debug)とは，プログラムに潜む誤りや欠陥を見つけ，バグを取り除きプログラムを修正する作業のことである。エラーが表示されたときは，このデバッグ作業を行う。

（１）おもなエラーの種類

①構文エラー(文法エラー)

スペルミスやインデントミスなど、Python のプログラムの構文として正しくない場合に発生するエラーを構文エラー(文法エラー)という。

②例外エラー(実行時エラー)

プログラムを正しく実行できなかったときに発生する。例えば，プログラムでファイルの読み込みを指示しているが，該当のファイルが指定した場所にないときは，プログラムが実行できず，実行時にエラーとなる。

③論理エラー

文法上の間違いはなく，またプログラムを実行してもエラー表示されないが，実行結果が思うようなものになっていないエラーを論理エラーという。この場合はプログラムを部分的に実行するなどを行ってエラーの原因を探す。

（2）Python のエラーメッセージ

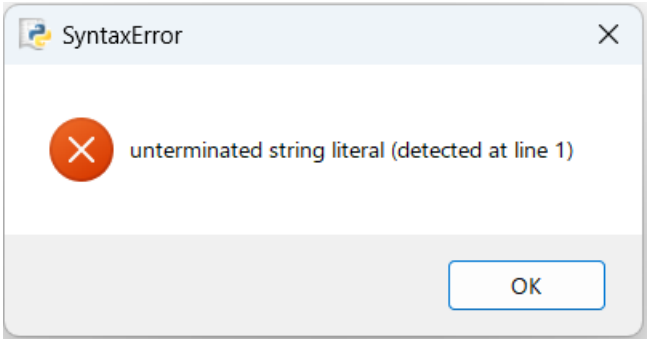
Python のプログラムを実行したときに表示される代表的なエラーメッセージには，次のようなものがある。

①シンタックスエラー(SyntaxError)

文法上の誤りがあるときに表示されるエラーメッセージである。

（例）

コード	解説
`print('こんにちは)`	'こんにちは'と入力するところ，誤って後方の'が欠落



②インデントエラー(IndentationError)

インデントが必要な箇所でインデントしていないときに表示されるエラーメッセージである。

（例）

コード	解説

`a = 10`
`b = 20`
`if a < b:`
`print('b が大きいです')`
	
	if 文
	条件が成立時の処理をインデントしないで記述

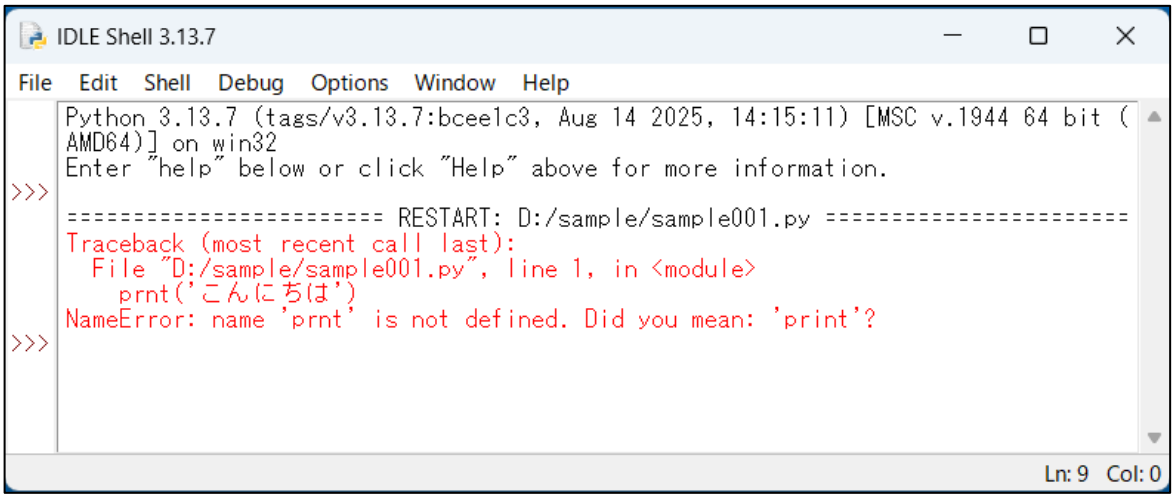
IDLE のウィンドウには SyntaxError として表示される。

③ネームエラー(NameError)

関数名などを誤って入力したときに表示されるエラーメッセージである。

(例)

コード	解説
`prnt('こんにちは')`	print を誤って prnt と入力



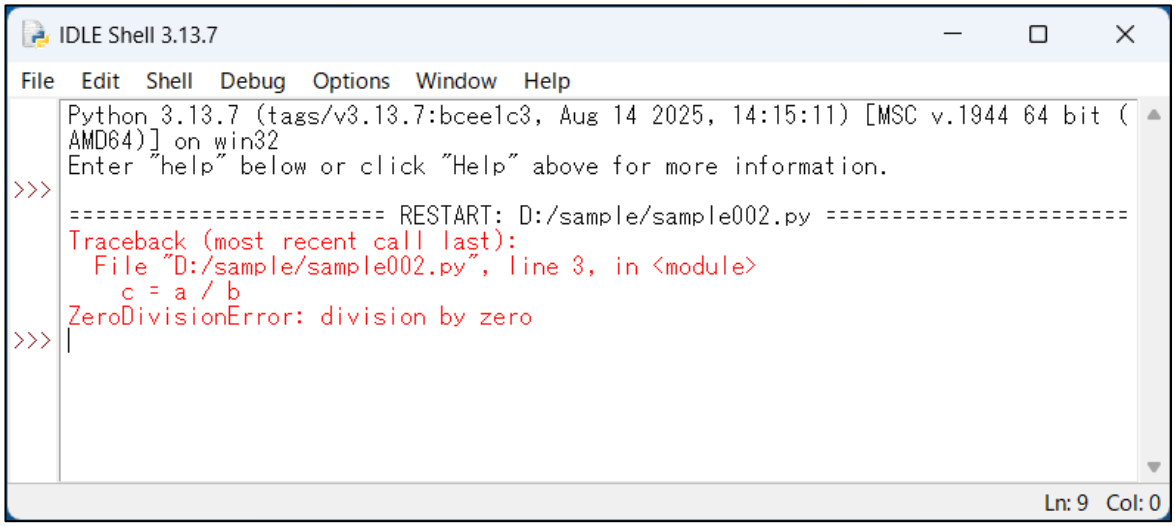
④ゼロ除算エラー(ZeroDivisionError)

プログラムの記述は文法的に正しいが，プログラムを実行すると発生する例外的なエラーがある。その一つに 0 での割り算がある。

(例)

コード	解説

`a = 10`
`b = 0`
`c = a / b`
`print(c)`
	0 での割り算



（3）無限ループを止める方法

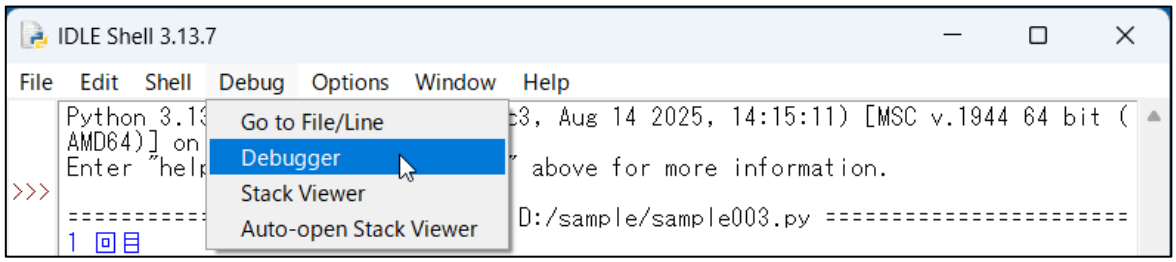
反復処理などを行うとき，処理が終わらず永遠に実行されてしまう状態を無限ループという。Windows では，キーボードから Ctrl キーを押したまま C を押すと無限ループのプログラムが終了する。

１５．IDLE のデバッグ機能

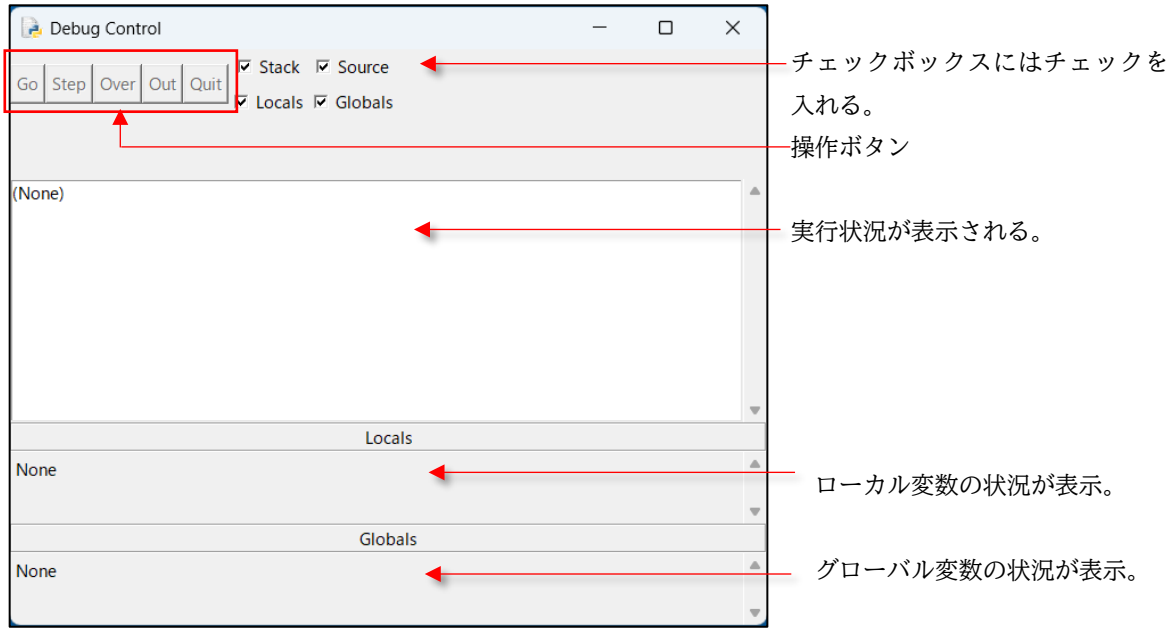
論理エラーの場合は，文法上の間違いがないためエラーメッセージが表示されない。そのため，プログラムを部分的に実行するなどの作業を行ってエラーの原因を探す必要がある。IDLE では，この作業を行うのに便利なデバッガーが用意されている。

（１）デバッガーの起動

①シェルウィンドウのメニューバーから[Debug]－[Debugger]をクリックする。



②デバッガーが起動し，[Debug Control]のウィンドウが表示される。



（２）操作ボタン

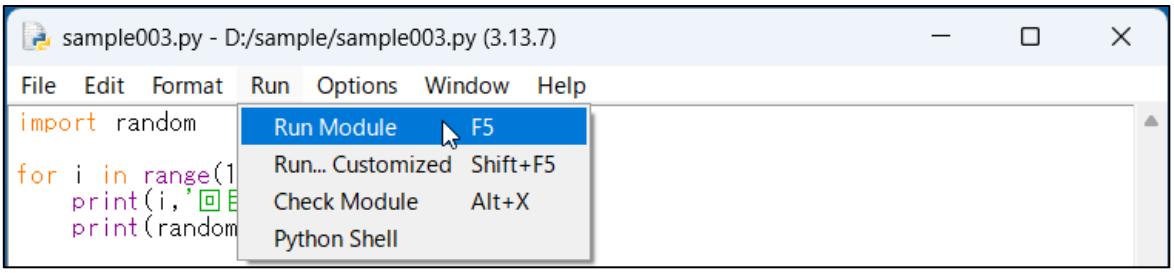
デバッグは次の五つのボタンを使って行う。

ボタン	内容
Go	ブレークポイントの直前までプログラムを実行する。ブレークポイントがないときは，最後までプログラムを実行する。
Step	現在の行にあるプログラムを実行し，次の行で停止する。現在の行に関数の呼び出しがあるとデバッガーはその関数の中に入る。関数のデバッグも行うときに使用する。
Over	現在の行にあるプログラムを実行し，次の行で停止する。現在の行に関数の呼び出しがあると，その関数を実行してから次の行で停止する。通常のデバッグ時に利用する。
Out	現在の関数を最後まで実行してから関数の外に移動する。Step で関数の中に入った場合，関数の外に出ることができる。
Quit	デバッグを終了する。

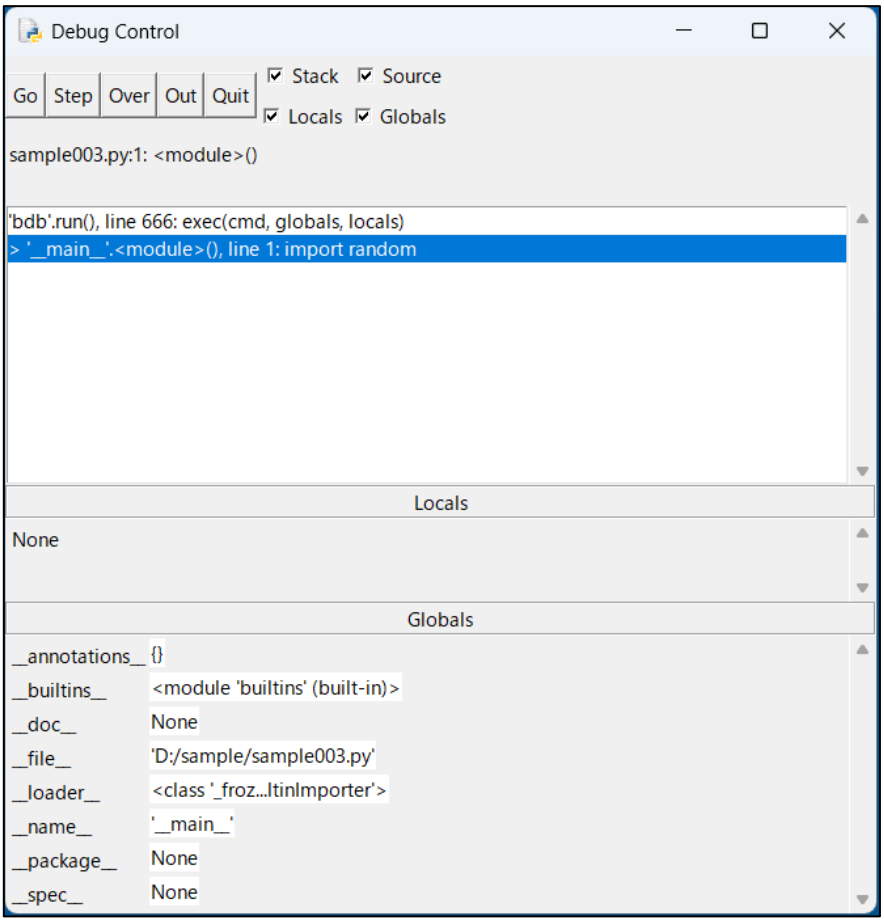
（3）ステップ実行

1行ずつプログラムを実行しながらチェックする方法が，ステップ実行である。

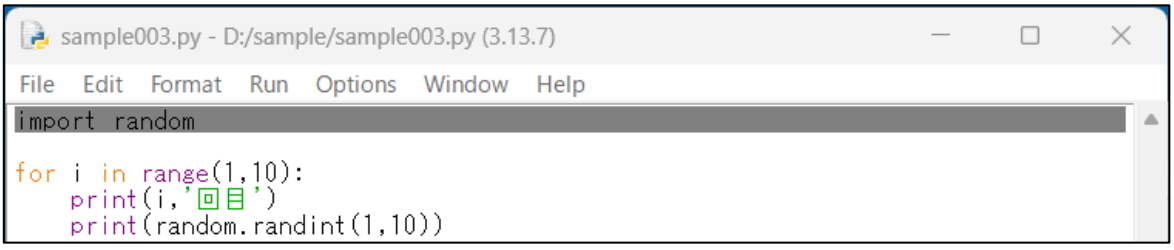
①[Debug Control]のウィンドウが表示された状態で，プログラムのエディタのメニューから[Run]-「Run Module」をクリックし，プログラムを実行する。



②[Debug Control]のウィンドウに1行目で停止していることが表示される。



③エディタウィンドウでは，停止している行が網掛けされて表示される。

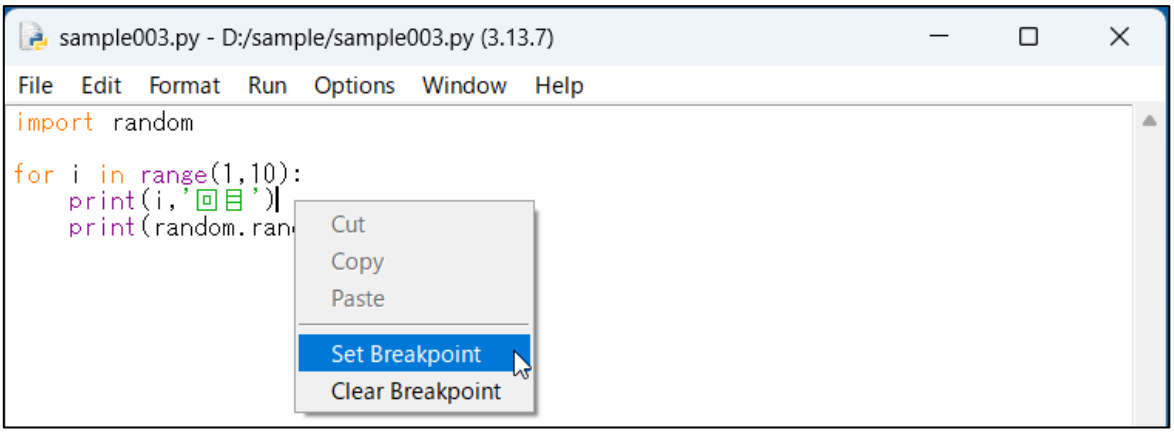


④操作ボタンを使ってステップ実行し，プログラムを確認する。

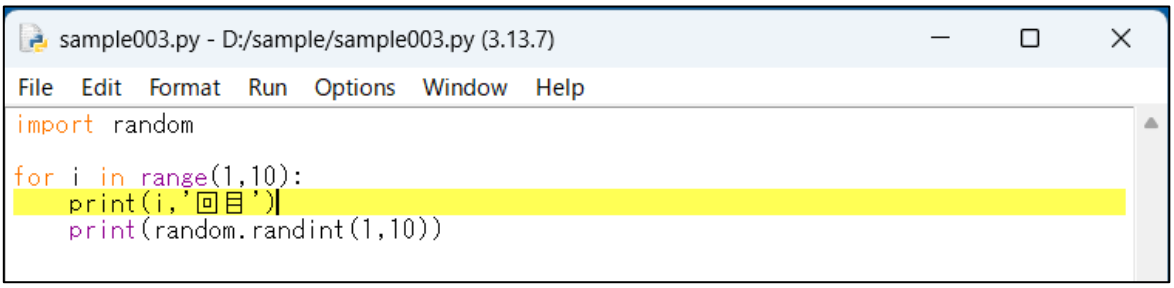
（4）ブレークポイント

エラーの箇所を特定するには，ブレークポイントを設定すると便利である。

①エディタウィンドウ上でブレークポイントを設定する行で右クリックし，[Set Breakpoint]を選択する。



②設定した箇所が黄色で表示される。



③[Debug Control]のウィンドウが表示された状態で，プログラムのエディタのメニューから[Run]-「Run Module」をクリックし，プログラムを実行する。

④操作ボタンを使ってステップを実行し，プログラムを確認する。

⑤ブレークポイントを解除するときは，ブレークポイントを設定した行で右クリックし，[Clear Breakpoint]を選択する。