

付録

A

「練習問題」
「この章の理解度チェック」
解答

第1章の解答

練習問題 1.1 p.10

- [1] Pythonはソースコードをそのまま実行できる**インタプリタ言語**（逐次翻訳言語）であり、トライ&エラーの作業を迅速にできます。オブジェクト指向言語を中心に、手続き型言語、関数型言語などの特徴も持つ**マルチパラダイム言語**で、プログラミング言語のさまざまな特徴を兼ね備えています。標準で**ライブラリ**（道具）も豊富にそろえており、Pythonをインストールするだけで高度なアプリを開発できる環境が整います。

この章の理解度チェック p.52

- [1] 作成するコードは、リストA.1の通りです。1.4.3項の内容は、今後もコードを作成&実行する中で欠かせないものです。同じ手順を何度も繰り返して、体になじませてください。

▶リストA.1 ex_hello.py

```
print('こんにちは、世界！')
```

- [2] 文は改行で区切ります（「;」などの区切り文字は不要です）。

文を途中で改行するには、行末に「\」（バックスラッシュ）を置きます。ただし、(...), [...], {...}などのカッコ内であれば、単語の区切りで自由に改行できます。

- [3] ● # : 単一行コメント
● '''...''', """...""" : 複数行コメント

一般的には、入れ子にできない複数行コメントよりも単一行コメントを利用すべきです。なお、複数行コメントは、正確には専用のコメント構文ではなく、複数行文字列の構文です。

- [4] 以下の3点を指摘できていれば正解です。

- 文字列はクォートでくる（山田は'山田'）
- 大文字／小文字は区別される（Nameはname）

- カッコの外のインデントは意味を持つので不可

以上の点を修正したコードが、リストA.2です。

▶リストA.2 ex_show_name.py

```
name = '山田'
print(name)
```

第2章の解答

練習問題 2.1 p.61

- ① 誤り。識別子を数字で始めることはできません（2文字目以降は可）。
- ② 正しい。変数は小文字のアンダースコア記法（スネークケース記法）が基本ですが、Pascal記法でも文法上の誤りではありません。
- ③ 正しい。識別子にはマルチバイト文字も利用できます。ただし、一般的には英数字、アンダースコアの範囲にとどめてください。
- ④ 誤り。予約語を識別子にはできません。
- ⑤ 誤り。「-」は演算子なので、識別子の一部として利用することはできません。

練習問題 2.2 p.72

- [1] 以下から3つ以上挙げられれば正解です。詳しくは表2.4（p.62）を参照してください。

- 数値 : 整数 (int)、浮動小数点 (float)、複素数 (complex)
- データ : 文字列 (str)、バイナリ (bytes)
- コンテナ : リスト (list)、タプル (tuple)、辞書 (dict)、セット (set)
- その他 : 論理型 (bool)、NoneType

- [2] 以下のようなリテラルを表現できていれば正解です。

- ① 0xff
接頭辞が0xで、以降の数値は0~9、a~f（A~F）であること。

② 123_456

アンダースコアで、一般的には3桁ごとに数値を区切る。先頭／末尾のアンダースコアは不可。

③ 'こんにちは、\nあかちゃん'

「\n」がエスケープシーケンスの改行。

④ 0.14142e-3

<仮数部>e<符号><指数部>の形式。

⑤ r'c:\data\hoge'

r'...'、r"..."の形式。配下の文字列ではエスケープシーケンスが無視されます。

この章の理解度チェック p.78

[1] ① 0o

② 0x

③ 仮数部

④ 指数部

⑤ エスケープシーケンス

⑥ \t

⑦ {...}

⑧ f

[2] 次のポイントを挙げられれば正解です。

- 変数の宣言には、データ型は不要です（ここではstr）。
- 文字列リテラルはシングルクォート（'）、またはダブルクォート（"）でくくります。
- 大文字小文字は区別されます（Printはprint）。

以上を修正したコードは、リストA.3の通りです。

▶リストA.3 ex_hello.py

```
msg = 'こんにちは、Python ! '
print(msg) # 結果：こんにちは、Python !
```

[3] (×) NoneはNoneType型で、bool型はTrue/Falseの2個です。

(×) raw文字列ではなく、複数行文字列(''''...''')を利用します。

(×) 一般的には、英数字とアンダースコアだけで構成すべきです。ただし、文法上は日本語を含むほとんどのUnicode文字を識別子として

利用できます。

(×) 異なる型の値を並べても間違いではありません（ただし、一般的には型をそろえるべきです）。

(○) 正しい。

[4] ① del name

② txt = 'みかん\tかき\tりんご'

```
③ data = [
    ['あ', 'い', 'う', 'え', 'お'],
    ['か', 'き', 'く', 'け', 'こ'],
]
```

④ print(f'こんにちは、{name}さん')

⑤ print(data[4])

⑤では、末尾のインデックスは「リストの個数 - 1」である点に注意してください。

第3章の解答

練習問題 3.1 p.84

[1] ① 45

② 16

③ 1

④ エラー

⑤ 2

'...'でくくられた数字はリテラルは文字列となるため、文字列の連結とみなされます(①)。また、「/」「//」「%」によるゼロ除算(④)は、エラー(ZeroDivisionError)となります。

[2] 浮動小数点数(float)は、内部的には2進数で演算されるためです。10進数の0.1も2進数では0.000110...（無限循環小数）となり、誤差の原因となります。演算誤差を防ぐには、Decimal型を利用してください。

練習問題 3.2 p.101

[1] リストA.4のようなコードが書けていれば正解です。

▶リストA.4 p_if.py

```
value = 'はじめてまして'
print('値なし' if value is None else value)
```

[2] ① True

② エラー

異なる型同士での大小比較はエラーになります。

③ False

文字列と整数値のように異なる型同士でも == 比較は可能ですが、False を返します。

④ False

リストの大小比較は、先頭から要素を大小比較し、最初に大小が決まったところで確定します。

この章の理解度チェック p.109

[1] ① 算術演算子（代数演算子でも可）

② 代入演算子

③ 複合代入演算子

④ 比較演算子

⑤ 論理演算子

⑥ &、^、|、~、<<、>> から3個以上

[2] x: 40、y: 50、data1、data2 ともに: [10, 15, 30]

ミュータブル（変更可能）な型とイミュータブル（変更不可）な型とで代入の挙動が異なる点に注意してください。イミュータブル型での代入は、常に値（オブジェクト）そのものの置き換えなので、代入元での変更が代入先に影響することはありません。

[3] ① 優先順位

② 結合則

③ 高い

④ 同じ

⑤ **

[4] ① i -= 2

② d = Decimal('0.5')

③ x, y, *z = [2, 4, 6, 8, 10]

④ n, m = m, n

⑤ 10 <= x < 50

decimal型を生成する際には、小数点数は文字列（クォート付き）で渡します（②）。

第4章の解答

練習問題 4.1 p.123

- [1] リストA.5のようなコードが記述できていれば正解です。if命令で多岐分岐を表現する場合には、条件式を記述する順番に要注意です。

▶リストA.5 p_if.py

```
point = 75

if point >= 90:
    print('優')
elif point >= 70:
    print('良')
elif point >= 50:
    print('可')
else:
    print('不可')
```

- [2] ペン図については、図4.7（p.123）を参照してください。このような置き換えルールをド・モルガンの法則と言います。法則を忘れてしまった場合にも、ペン図で理解しておくことで、自分で置き換えが可能となります。

練習問題 4.2 p.140

- [1] リストA.6のようなコードが書けていれば正解です。

▶リストA.6 p_for.py

```
for i in range(1, 10):
    for j in range(1, 10):
        result = i * j
        print(result, end=' ')
    print()
```

このように、制御命令は入れ子にできます。その場合、for命令の仮変数は、それぞれで異なる名前でないといけない点に注意してください。

練習問題 4.3 p.149

- [1] スキップ : continue 命令
脱出 : break 命令

continue、breakはいずれもfor、whileなどの繰り返し構文の中で使用できる命令です。その性質上、continue、break命令はif命令などの条件分岐と組み合わせて利用するのが一般的です。

[2] リストA.7のようなコードが書けていれば正解です。

▶リストA.7 p_while.py

```
i = 0
sum = 0

while i <= 100:
    i += 1
    if i % 2 != 0:
        continue
    sum += i

print(f'合計値は{sum}です。')
```

while命令では終了条件を表す変数iを、自分でインクリメント（カウントアップ）しなければなりません。ここでは頭の体操としてwhile命令を利用しましたが、変数をインクリメントするようなループではfor + rangeを利用するほうがコードはシンプルになります。

この章の理解度チェック p.156

- [1] ① if
- ② while
- ③ 節
- ④ ヘッダー
- ⑤ ブロック（スイート）
- ⑥ コロン（:）
- ⑦ インデント

複合文に関する基本的な問いです。キーワードを覚えることは本質的な目的ではありませんが、キーワードを整理することで、複合文の構造を復習してください。

[2] ① for item in data:
 print(item)

- ② for i in range(1, 101):
 print(i)
- ③ data2 = [i * 10 for i in data]
- ④ result = sum([i for i in data if i >= 0])
- ⑤ if 10 <= num < 50:
 print(num)

⑤の条件式は「num >= 10 and num < 50」としてもほぼ同じ意味ですが、回答例のコードのほうが範囲であることを把握しやすく、処理効率の観点から有利です。

- [3] ① while
- ② try
 - ③ num
 - ④ break

「while True:～」のようにすることで、意図して無限ループを作成することもできます。その場合は、④のように自前でループ脱出の出口を用意してください。

[4] リストA.8のようなコードが書けていれば正解です。

▶リストA.8 ex_for.py

```
sum = 0

for i in range(100, 201):
    if i % 2 == 0:
        continue
    sum += i
print(f'合計値は{sum}です。')
# 結果：合計値は 7500 です。
```

ここではcontinueを使ってとあるので、このようにしていますが、その制限を無視すれば、range関数でstep（増分）を指定する方法もあります。

- [5] リスト A.9 のようなコードが書けていれば正解です。

▶リスト A.9 ex_if_lang.py

```
language = 'Python'

match language:
    case 'Python' | 'Perl' | 'Ruby':
        print('インタプリタ言語')
    case 'C#' | 'C++' | 'Java':
        print('コンパイル言語')
    case _:
        print('不明')
```

match 命令の OR パターンです。

別解として、if 命令 + in 演算子 (3.3 節) を利用して以下のように表してもかまいませんが、一般的には match 命令を利用したほうが素直でしょう。

▶リスト A.9 ex_if_lang2.py

```
language = 'Python'
interpreter = ['Python', 'Perl', 'Ruby']
compiler = ['C#', 'C++', 'Java']

if language in interpreter:
    print('インタプリタ言語')
elif language in compiler:
    print('コンパイル言語')
else:
    print('不明')
```

第 5 章の解答

練習問題 5.1 p.167

- [1] 「値，…」のように値を列挙するだけの引数を「位置引数」、「名前=値，…」形式の引数を「キーワード引数」と呼びます。たとえば、以下の print 関数であれば、最初の「Hello', name, 'さん!」が位置引数で、以降の sep/end がキーワード引数です。

```
print('Hello', name, 'さん!', sep='+', end='')
```

- [2] メソッドとは、型にひもづいた、その型からのみ呼び出せる関数です。呼び出しには「インスタンス.メソッ

ド名(引数, …)」または「型名.メソッド名(引数, …)」と表します。

練習問題 5.2 p.191

- [1] リスト A.10 のようなコードが書けていれば正解です。

▶リスト A.10 p_slice.py

```
data = 'プログラミング言語'
print(data[4:7])
```

スライス構文は、インデックス値が 0 スタートで、開始位置～終了位置 - 1 文字目の文字列を抜き出します。

- [2] リスト A.11 のようなコードが書けていれば正解です。「\t」はエスケープシーケンスの一種でタブ文字を表します。

▶リスト A.11 p_split.py

```
data1 = '鈴木\t太郎\t男\t50歳\t広島県'
data2 = data1.split('\t')
print('&'.join(data2))
```

練習問題 5.3 p.199

- [1] リスト A.12 のようなコードが書けていれば正解です。

▶リスト A.12 p_datetime.py

```
from datetime import datetime

dt = datetime.now()
print(dt.month)
print(dt.minute)
```

この章の理解度チェック p.205

- [1] ① インスタンス化
② インスタンス (またはオブジェクト)
③ date
④ メソッド
⑤ .

クラス（型）の利用に関する問いです。インスタンスを生成し、メソッド／属性にアクセスする流れを、関連するキーワードとともに整理しておきましょう。

- [2] ① title[2]
② title[2:5]
③ title[2:]
④ title[-7:-5]
⑤ title[::2]

スライス構文を利用することで、文字列／リストの任意の位置から部分的な文字列／リストを取り出せるようになります。よく利用する構文なので、基本的な記述を再確認しておきましょう。

- [3] ① setlocale
② ja_JP.UTF-8
③ timezone
④ timedelta
⑤ strftime

datetime モジュールによる日時の生成から整形までの流れを問う問題です。書式文字列に日本語が含まれる場合には、ロケールも日本語（ja_JP.UTF-8）としておきます。

- [4] それぞれ、以下のようなコードを書けていれば正解です。

- ① data = 'となりのきやくはよくきやくくうきやくだ'
print(data.rfind('きやく'))
- ② locale = '千葉'
temp = 17.256
print('{0}の気温は、{1:.2f}℃です。'.format(locale, temp))
- ③ intro = '彼女の名前は花子です.'
print(intro.replace('彼女', '妻'))
- ④ from datetime import datetime, timedelta

```
dt = datetime.today()
dt_p = dt + timedelta(days=5, hours=6)
```

- ⑤ import calendar
calendar.setfirstweekday(6)
print(calendar.month(2025, 10, 5))

②はフォーマット文字列を利用して、以下のように表示してもかまいません。

```
print(f'{locale}の気温は、{temp:.2f}℃です。')
```

第6章の解答

練習問題 6.1 p.239

- [1] ① 可能
② 持たず
③ 不可
④ 辞書型（マッピング型も可）
⑤ キー／値

コレクションの大分類を理解する問いです。個々の用法を理解するだけでなく、型の特徴を知り、適材適所を使い分けていける能力を身につけてください。

- [2] ① pop
② append
③ insert
④ for
⑤ enumerate

最終結果からリストへの操作を類推する問題です。リストの基本的な役割を思い出してみましょう。

練習問題 6.2 p.247

- [1] リスト A.13 のようなコードを書けていれば正解です。

```
▶ リスト A.13 p_intersection.py
sets1 = {10, 105, 30, 7}
sets2 = {105, 28, 32, 7}
print(sets1.intersection(sets2))
```

set型は、このような集合演算で力を発揮します。intersectionメソッドの代わりに「&」演算子を利用してかまいません。

この章の理解度チェック p.260

- [1] (×) 要素の挿入／削除は、要素の移動を伴うため、先頭に近くなるほど遅くなります。
- (×) dequeでは、位置に関わらず、要素の挿入／削除は高速です。ただし、挿入／削除に先立って、要素の検索が加わるはずなので、実際には、そちらのオーバーヘッドを考慮しなければなりません（中央に近づくほど、アクセスは低速になります）。
- (×) セット型は、要素の並び順を管理しません。
- (×) 組み込み型では、int、str、tuple、frozensetなどが辞書のキーにできますし、一定の条件を満たしていればユーザー定義型をキーに指定することも可能です。
- (×) スタックとキューとは逆の記述になっています。

コレクションはそれぞれに得手不得手を持っています。用法そのものはいずれの型もほぼ共通しているので、その時どきの用途に応じて、適切な型を使い分けるのが重要です。

- [2] ① cucumber
② 胡瓜
③ pop
④ setdefault
⑤ item
⑥ d.items()

最終結果から辞書への操作内容を類推する問題です。インデックス構文をはじめ、pop/setdefaultなど、基本的な辞書の操作を再確認してください。④のsetdefaultメソッドは、この場合であれば、インデックス構文を利用して「d['carrot'] = 'ニンジン」としてもかまいません。

- [3] ● 値をカンマで区切ってセットを生成するには[...]

ではなく、{...}でくくります。

- differenceはunionの誤りです。
- whileはforの誤りです。

以上を修正したコードは、リストA.14の通りです。

▶リストA.14 ex_set.py

```
sets1 = {2, 4, 8, 16, 32}
sets2 = {1, 10, 4, 16}

print(sets1.union(sets2))
sets3 = {str(i) for i in sets1 if i > 5}
print(sets3)
```

- [4] ① print(d.get('apple', '-'))
② data = [elem for elem in data if elem != '×']
③ data[0:3] = []
④ t = ('いろは',)
⑤ for key, value in d.items():
 print(f'{key}={value}')
- ②は「×」でない要素を取り出し、新たなリストを生成する、という意味です。これで結果として「×」がすべて削除されます。
- ③は「del data[0:3]」としても正解です。
- ④のように、一要素のタプルを生成する場合には、末尾のカンマは必須です。

第7章の解答

練習問題 7.1 p.283

- [1] リストA.15のようなコードが書けていれば正解です。

▶リストA.15 p_search.py

```
import re

data = '住所は〒160-0000 新宿区南町0-0-0②  
です。\\nあなたの住所は〒210-9999 川崎市北町③  
1-1-1ですね'
```

```
ptn = re.compile(r'\d{3}-\d{4}')
results = ptn.finditer(data)
for result in results:
    print(result.group())
```


最初からマッチしている文字列が1つとわかっている場合には、searchメソッドを利用してもかまいません。

[2] リストA.16のようなコードが書けていれば正解です。

▶リストA.16 p_group.py

```
import re

data = 'お問い合わせはsupport@example.comまで'

ptn = re.compile(r'[a-z0-9.!#$%&\'*+/?^_{}~]+@[a-z0-9-]+(?:\. [a-z0-9-]+)*',
                 re.IGNORECASE)
print(ptn.sub(r'<a href="mailto:\g<0>">>\g<0></a>', data))
```

マッチした文字列を置換後の文字列に反映させるには、特殊変数として\g<0>を利用します。サブマッチ文字列を引用するならば\g<1>、\g<2>...を利用します。

練習問題 7.2 p.299

- [1] ① csv
② with
③ w
④ delimiter
⑤ writerows

csvモジュールを利用した場合も、withブロックでファイルを開閉し、writexxxxxメソッドで書き込む流れは、一般的なファイル操作の場合と共通です。読み込みの場合も復習しておきましょう。

練習問題 7.3 p.313

- [1] ① requests
② get
③ status_code

requestsモジュールによるネットワーク通信の例です。HTTP GETによる基本的な流れを理解できてしまえば、HTTP POST、JSONなどによる通信も同

じように記述できます。

この章の理解度チェック p.322

- [1] ① .
② ?i、?m、?s、?x、?L から2個以上
③ +?、*?、??、{n,}? などから2個以上
④ ?P<name>
⑤ A(?=B)

正規表現そのものに関する問いです。Pythonとは直接の関係はありませんが、コーディングの幅を広げる知識です。実際に文字列をマッチさせながら、表現の引き出しを増やしていきましょう。

- [2] ① re
② compile
③ line
④ finder
⑤ result.group()

正規表現とファイル読み込みの複合問題です。間違ってしまったという人は、7.1節、7.2節を見直してみましょう。

[3] それぞれ、以下のようなコードを書けていれば正解です。

```
① print(abs(-12))
② print(round(987.654, 2))
③ import os
   for f in os.listdir('.'):
       print(f)
④ import random
   print(random.randint(0, 100))
⑤ import re
   ptn = re.compile(r'[/\\]')
   result = ptn.split(txt)
```

第8章の解答

練習問題 8.1 p.332

- [1] リストA.17のようなコードが書けていれば正解です。

▶リストA.17 p_func.py

```
def get_diamond(diagonal1, diagonal2):  
    return diagonal1 * diagonal2 / 2  
  
area = get_diamond(8, 10)  
print(f'菱形の面積は{area}です。')  
# 結果: 菱形の面積は40.0です。
```

練習問題 8.2 p.351

- [1] ユーザー定義関数の外で定義された変数のことを「グローバル変数」、関数内で定義された変数のことを「ローカル変数」と言います。グローバル変数はコード（ファイル）全体から参照できますが、ローカル変数は関数内でしか参照できません。
- [2] defブロックの中で、global宣言を使ってグローバル変数であることを明示的に宣言します。global宣言がない場合、関数内での代入操作はすべてローカル変数に対するものとみなされます。

練習問題 8.3 p.363

- [1] リストA.18のようなコードが書けていれば正解です。

▶リストA.18 p_args_param.py

```
def average(*values: float) -> float:  
    result = 0  
    for value in values:  
        result += value  
    return result / len(values)  
  
print(average(5, 7, 8, 2, 1, 15))  
# 結果: 6.333333333333333
```

この章の理解度チェック p.378

- [1] (×) return命令がない場合、関数はNoneを返したとみなされます。
(×) Pythonのグローバルスコープは、いわゆるファイル（モジュール）スコープです。ファイルの中でのみ参照できます。
(○) 正しい。
(×) 可変長引数の実体はタプルです。args[0]のようにアクセスします。
(×) 「初回呼び出し」は「定義」の誤りです。
- [2] リストA.19のようなコードが書けていれば正解です。

▶リストA.19 ex_args_default.py

```
def get_square(base: float = 1.0, height: float = 1.0) -> float:  
    return base * height  
  
print(f'平行四辺形の面積は{get_square()}です。')  
# 結果: 平行四辺形の面積は1.0です。  
print(f'平行四辺形の面積は{get_square(10)}です。')  
# 結果: 平行四辺形の面積は10.0です。  
print(f'平行四辺形の面積は{get_square(10.5, 5)}です。')  
# 結果: 平行四辺形の面積は52.5です。
```

ユーザー定義関数の基本的な構文を問う問題です。引数の既定値は「仮引数=値」の形式で表します。

- [3] ① Callable[[int], int]
② *nums
③ list
④ for
⑤ appned
⑥ func(num)
⑦ lambda

可変長引数、高階関数、ラムダ式に関する複合的な問題です。ラムダ式は「lambda 引数: 本体」の形式で表します。

- [4] ① 110
 ② 100
 ③ [100, 20, 30]
 ④ [100, 20, 30]

引数がミュータブル型（ここではlist）の場合、引数への変更操作は呼び出し元にも影響します。イミュータブル型（ここではint）では、引数の変更はそのままオブジェクトそのものの置き換えなので、呼び出し元への影響はありません。

第9章の解答

練習問題 9.1 p.400

- [1] ① Generator
 ② random
 ③ Generator[int]
 ④ yield
 ⑤ max
 ⑥ 100
 ⑦ break

random_intは無限に値を生成するジェネレーターです。このようなジェネレーターを利用する場合には、呼び出し元でなんらかの終了条件（ここでは「num > 80」）を設ける必要があります。

練習問題 9.2 p.416

- [1] ① from math import abs, ceil, floor
 ② __all__ = ['hoge', 'piyo']
 ③ sys.path.append('c:/data')
- ②の__all__変数は、モジュールファイルの冒頭で記述するのが一般的です。

練習問題 9.3 p.431

- [1] コルーチンとは、任意の地点で中断でき、あとから再開できる性質を持つ関数の一種です。defブロック

にasyncキーワードを付与することで定義できます。中断のポイントは、await演算子で表します。

この章の理解度チェック p.440

- [1] (○) 正しい。
 (×) 「@名前(...)」の誤り。
 (×) ジェネレーターを呼び出した場合、Generatorオブジェクトが返されるだけです。
 (×) イベントループはあくまでI/O待ちで複数の処理を切り替え、実行するための仕組みです。よって、CPUを占有するような処理では高速化を期待できません。
 (×) docstringは、def/class命令の直後（ブロックの先頭）に記述します。

- [2] ① now()
 ② func(*args, **kwargs)
 ③ result
 ④ inner
 ⑤ @time_deco

デコレーターでは、内側の関数が「付加すべき機能と本来の関数の呼び出し」を担い、外側の関数は内側の関数を返します。引数付きのデコレーターでは、さらに「引数を受け取る関数」が外側に加わるので混乱しがちですが、まずは入れ子となった関数の定型的な記述を覚え込んでしまいましょう。

- [3] 以下の点が指摘できていれば正解です。
- asyncとawaitの位置が逆
 - await式に渡せるのはtime.sleepではなくasyncio.sleep
 - フォーマット文字列の接頭辞は「r」ではなく「f」
 - 非同期処理のエントリーポイントはrun関数で呼び出す
 - コルーチンを束ねるのはrunではなくgather

以上を反映させたコードがリストA.20です。

▶リストA.20 ex_coroutine.py

```
import asyncio
import time

async def heavy(name: str, sec: int) -> str:
    print(f'{name} start...')
    await asyncio.sleep(sec)
    print(f'{name} end...')
    return f'{name}: {sec} sec'

async def main() -> None:
    print(await asyncio.gather(
        heavy('hoge', 2),
        heavy('bar', 5),
        heavy('piyo', 1),
    ))

start = time.time()
asyncio.run(main())
end = time.time()
print(f'Process Time: {end - start}')
```

第10章の解答

練習問題 10.1 p.464

[1] 以下の5点が指摘できていれば正解です。

- 初期化メソッドの名前は__new__ではなく__init__
- showメソッド定義の引数にselfが抜けている
- 直接にコードが実行されたかを判断するには、__name__が__main__であることを確認する(__app__ではない)
- クラスをインスタンス化するには「クラス名(...)」(newは不要)
- メソッド呼び出しは「->」ではなく「.」

以上を修正した正しいコードは、リストA.21の通りです。

▶リストA.21 p_class.py

```
class Pet:
    def __init__(self, kind: str,
name: str) -> None:
        self.kind = kind
        self.name = name

    def show(self) -> None:
        print(f'ペットの{self.kind}の名前は、
{self.name}ちゃんです！')

if __name__ == '__main__':
    p = Pet('ハムスター', 'のどか')
    p.show()
```

練習問題 10.2 p.494

- [1] ① 基底クラス（スーパークラス、親クラスでも可）
 ② 派生クラス（サブクラス、子クラスでも可）
 ③ (Person)

派生クラスは、「class 派生クラス名(基底クラス名,...):」の形式で定義します。基底クラスは必要に応じて複数列举することも可能です。

[2] リストA.22のようなコードが書けていれば正解です。

▶リストA.22 p_inherit.py

```
from typing import override

class MySubClass(MyClass):
    @override
    def show(self) -> str:
        return f'[{super().show()}]'

if __name__ == '__main__':
    ms = MySubClass('ハムスター', 'のどか')
    print(ms.show())
    # 結果: [ペットのハムスターの名前は、
    のどかです。]
```

この章の理解度チェック p.503

- [1] ① __init__（初期化メソッドでも可）
 ② インスタンス変数

- ③ クラスメソッド
- ④ @classmethod
- ⑤ cls
- ⑥ is-a
- ⑦ オーバーライド
- ⑧ super
- ⑨ 委譲
- ⑩ ミックスイン

オーバーライドはオーバーロードと間違えないように注意してください。

- [2] (×) 実際には、内部的にリネームしているだけなので完全に隠蔽できるわけではありません。
- (×) 抽象クラスはミックスインの誤り。抽象クラスは派生クラスでオーバーライドすべきメソッド（抽象メソッド）を含んだクラスです。
- (×) @prop.setter／@prop.getterデコレーターは、@property／@<名前>.setterデコレーターの間違いです。
- (×) 継承は基底クラスと強い結びつきを持つため、変更に対する修正コストが高まる可能性があります。継承よりも委譲を優先すべきです。
- (×) isinstance関数は、指定されたクラスの直接のインスタンス、または派生クラスのインスタンスである場合にTrueを返します。

- [3] 次の点を指摘できれば、正解です。

- FigureクラスはabcモジュールのABCクラスを継承する必要があります。
- @abstractデコレーターは、@abstractmethodデコレーターの間違いです。
- クラスを継承するには、extendsは使わず、派生クラスの定義で引数に基底クラスを指定します(2か所)。
- issubclassはisinstanceの誤りです。
- __name__は__class__の誤りです。

修正済みのex_polymo.pyについては、リストA.23の通りです。

▶リストA.23 ex_polymo.py

```
from abc import abstractmethod, ABC

class Figure(ABC):
    def __init__(self, width: float, height: float) -> None:
        self.width = width
        self.height = height

    @abstractmethod
    def get_area(self) -> float:
        pass

class Triangle(Figure):
    def get_area(self) -> float:
        return self.width * self.height / 2

class Square(Figure):
    def get_area(self) -> float:
        return self.width * self.height

if __name__ == '__main__':
    figs = [
        Triangle(10, 15),
        Square(10, 15),
        Triangle(5, 1)
    ]
    for fig in figs:
        if isinstance(fig, Figure):
            print(f'{fig.__class__}: {fig.get_area()}')
```

- [4] リストA.24のようなコードができていれば正解です。

▶リストA.24 ex_class.py

```
class Hamster:
    # インスタンス変数を初期化
    def __init__(self, name: str) -> None:
        self.__name = name

    # 読み取り専用のnameプロパティ
    @property
    def name(self) -> str:
        return self.__name

    # 与えられた書式を使って__nameの値を出力
    def show(self, fmt: str) -> None:
        print(fmt.format(self.__name))

if __name__ == '__main__':
    h = Hamster('のどか')
    h.show('私の名前は{0}です！')
```

第11章の解答

練習問題 11.1 p.525

[1] より下位の例外クラスを先に記述します。exceptブロックは先に書かれたものが優先されるため、たとえば Exception クラスを最初に記述した場合には、すべての例外がそこで捕捉されてしまい、以降のexceptブロックが呼び出されることはありません。

[2] 以下のような点を説明できていれば正解です。

- 具体的な例外の内容を識別できるよう、汎用的な Exception 例外の発生は避ける。
- 標準的な例外が用意されているものは、独自例外よりも標準例外を利用する。
- その場で処理できない例外は握りつぶすのではなく、raise を使って現在の例外をそのまま呼び出し元に再送出する。

練習問題 11.2 p.548

- [1] ① class
② __init__
③ self
④ __eq__
⑤ other: object
⑥ isinstance
⑦ and \
⑧ False

クラスの基本的な定義と __eq__ メソッドの典型的な実装を問う問題です。忘れてしまった人は、第10.1節 (p.412) と11.2.2項 (p.482) を再度確認しておきましょう。

練習問題 11.3 p.560

[1] リストA.25のようなコードが書けていれば正解です。

▶ リストA.25 p_dataclass.py

```
import dataclasses

@dataclasses.dataclass(frozen=True)
class Hamster:
    name: str = '名無し'
    age: int = 0

    def show(self) -> None:
        print(f'{self.name}は{self.age}歳です!')

if __name__ == '__main__':
    h = Hamster('のどか', 1)
    h.show()
```

データクラスをイミュータブルにするには frozen パラメーターを指定するだけです。

この章の理解度チェック p.595

[1] (×) except ブロックは、発生した例外がexceptブロックのそれと一致、または派生クラスである場合に呼び出されます。

(×) finally ブロックの return を優先します。

(×) 逆の記述。捕捉の対象が明確になるように、できるだけ下位の例外を指定すべきです。

(○) 正しい。

(×) __iter__ はイテレーターを取得するためのメソッドです。イテレーターは __next__ メソッドを実装します。

[2] ① type

② __init__

③ __call__


④ cls.__instance is None

⑤ super()

⑥ metaclass=SingletonMeta

メタクラスでは、クラス定義になんらかの変数/メソッドを追加するならば __init__ メソッドを、実クラスのインスタンスをカスタマイズするならば __call__ メソッドをオーバーライドします。

[3] 以下のようなコードが書けていれば正解です。

```
① class Person:
    def __init__(self, name: str) -> 
None:
    self.name = name

    def __str__(self) -> str:
        return f'Person: {self.name}'

② class MyAppError(Exception):
    pass

③ import dataclasses
    @dataclasses.dataclass(frozen=True)
    class Book:
        title: str
        price: int

④ for f in dataclasses.fields(Book):
    print(f)

⑤ try:
    raise KeyError
except KeyError:
    raise
except TypeError:
    raise
finally:
    print('終了しました。')
```