日商プログラミング検定 STANDARD（Python）
セット3 詳細解答・文法解説

元の問題文は変更せず、解答に付随するPython文法の名称、背景、複数の実行例、注意点を追加した学習用解説です。実行例は原則Python 3を想定します。

知識科目・第1問（4択20問）

問1 ② def

1. 正解：② def 解説：Python で関数を定義するキーワードは「def」。 例：def my_func(): のように書く。func や function、method は キーワードではない。

【文法の理解】関数定義と呼び出し：def
は関数定義の予約語。関数名、丸括弧、コロン、インデントされた本体が必要。return は呼び出し元へ値を返す。

```
def add(a, b):
    return a + b
print(add(2, 3))    # 5
```

問2 ③ 2

2. 正解：③ 2 解説：Python の「%」演算子（剰余）は、除数（この場合3）と同じ符号 の結果を返す。-7 を 3 で割ると商は -3（切り捨て方向）で、 -7 - (3 × -3) = -7 - (-9) = 2 となる。よって -7 % 3 = 2。

【文法の理解】切り捨て除算と剰余：/ は通常の割り算、// は負の無限大方向へ丸める床除算、% は剰余。a == (a//b)*b + (a%b) が成立する（b≠0）。負数では単なる「小数点以下の削除」と異なる。

```
print(7 / 2)      # 3.5
print(7 // 2)     # 3
print(-7 // 3)    # -3
print(-7 % 3)     # 2
```

問3 ③ '''文字列'''

3. 正解：③ '''文字列''' 解説：Python で複数行にわたる文字列（トリプルクォート文字列）は、
シングルクォート3つ（'''）またはダブルクォート3つ（"""）で 囲んで表す。

【文法の理解】コメントと文字列：# から行末まではコメント。文字列内の #
は文字データである。三連引用符は複数行文字列を表し、複数行コメント専用の構文ではない。

```
# 行コメント
name = "A#B"
text = """1行目
2行目"""
print(text)
```

問4 ① +

4. 正解：① + 解説：文字列の連結（結合）には「+」演算子を使う。 例：'Hello' + 'World' → 'HelloWorld'

【文法の理解】演算子の使い分け：文字列同士の + は連結、数値同士の + は加算。/ は除算、// は床除算、% は剰余、** は累乗。型によって演算子の働きが変わる。

```
print("Py" + "thon") # Python
print("ha" * 3) # hahaha
print(8 // 3, 8 % 3) # 2 2
```

問5 ① for

5. 正解：① for 解説：for は Python の予約語（キーワード）であり、繰り返し構文のためにに予約されているため変数名として使用できない。for1、For、 f_or はキーワードと完全には一致しないため使用可能 （For は大文字小文字を区別するため for とは別の名前として有効）。

【文法の理解】予約語と大文字小文字：for、class、def などは予約語。for1 や For は別名として使用できるが、組み込み関数名を変数で上書きするのは避ける。

```
for1 = 3
For = 4
print(for1 + For) # 7
```

問6 ② 7点

6. 正解：② 7点 解説：f-string（f'...'）では、{ } で囲まれた部分がPythonの式として 評価される。{3+4} は 7 と計算され、「7点」と表示される。

【文法の理解】f-stringの式展開：f または F を付けた文字列では {式} を計算して埋め込む。書式指定は {値:.2f} 等を用いる。文字通りの波括弧は {{ と }}。

```
x = 7 / 3
print(f"結果は{x:.2f}点") # 結果は2.33点
print(f"{{x}} = {x}")
```

問7 ② (３)と(４)

7. 正解：② (３)と(４) 解説：Python の and/or は「最後に評価した値」を返す。 (1) 5 and 0 → 5は真なので0を返す → 0 (2) 0 and 5 → 0は偽なので0を返す → 0 (3) 5 or 0 → 5は真なので5を返す → 5 (4) 0 or 5 → 0は偽なので5を返す → 5 結果が5になるのは(3)と(4)。

【文法の理解】真偽値と短絡評価：and は左が偽なら左を返し、真なら右を返す。or は左が真なら左を返し、偽なら右を返す。必ずしも bool を返さない（not は bool を返す）。右辺が評価されない場合を短絡評価という。

```
print(2 and 0) # 0
print(0 or 2) # 2
print(not False) # True
print("" or "未入力") # 未入力
```

問8 ② 3 == 3.0

8. 正解：② 3 == 3.0 解説：Pythonでは整数と浮動小数点数を値として比較でき、 3 == 3.0 は True になる。①は型が異なる（int と str）ため False、③はリストとタプルは型が違うため同じ中身でも False、④はPythonの文字列比較が大文字小文字を区別するため False。

【文法の理解】等値比較と型：== は値の等しさ、is は同一オブジェクトかを比較する。整数と浮動小数点数は数値として等値になりうるが、文字列と数値は別物。

```
print(3 == 3.0) # True
print(3 == "3") # False
print([1,2] == (1,2)) # False
```

問9 ② B

9. 正解：② B 解説：score=85のとき、score >= 90 は False。 elif score >= 70 は True（85は70以上）なので print('B') が 実行される。

【文法の理解】if / elif / else：条件は上から評価され、初めて真となった枝のみが実行される。else は前の条件がすべて偽の場合。比較は >、>=、== などを使い、= は代入。

```
score = 85
if score >= 90:
        print("A")
elif score >= 70:
        print("B")
else:
        print("C")
```

問10 ③

10. 正解：③ 解説：「data が空でなければ『データあり』と表示、空なら何もしない」 という仕様に対して、③「if not data: print('データあり')」は リストが空のときに『データあり』と表示してしまい、条件が逆になっているため誤り。①②④は空でないリストのときだけ 「データあり」と表示する、同じ動作になる。

【文法の理解】空のリストの真偽：空のリスト [] は偽、要素のあるリストは真。if data: は if len(data)>0: と同様の判定。if not data: は空のとき成立する。

```
for data in [[], [0], [1,2]]:
        print(bool(data)) # False, True, True
```

問11 ② 6

11. 正解：② 6 解説：これは階乗（3の階乗）を求めるプログラム。 n=3: result=1*3=3, n=2 n=2: result=3*2=6, n=1 n=1: result=6*1=6, n=0（ここでループ終了） 最終的に result は 6（=3!）になる。

【文法の理解】階乗と累積積：積を求めるときは初期値1とし、掛け算を繰り返す。n! = 1×2×…×n。和の集計なら初期値0が基本。

```
result = 1
for n in range(1, 4):
        result *= n
print(result) # 6
```

問12 ① 15

12. 正解：① 15 解説：range(1, 6) は 1,2,3,4,5 を生成する。これらをすべて足すと 1+2+3+4+5=15。

【文法の理解】range(start,stop,step)：start から始まり、stop を含まず、step ずつ増減する。step を省略すると1。負の step では降順。print の end を使えば横並びにできる。

```
print(list(range(2,10,3))) # [2,5,8]
print(list(range(5,0,-1))) # [5,4,3,2,1]
print(list(range(1,6))) # [1,2,3,4,5]
```

問13 ③ エラーとなる（IndexError）

13. 正解：③ エラーとなる（IndexError） 解説：L1 = [1,2,3] の有効なインデックスは 0,1,2 のみ。 L1[3] は存在しない位置を指定しているため IndexError が発生する。

【文法の理解】スライス（部分列の取り出し）：リストの一部を取り出す構文を「スライス（slicing）」という。基本形は
seq[開始:終了:刻み幅]。開始は含み、終了は含まない。省略時は先頭・末尾・刻み幅1となる。結果は新しいリスト。

```
L = [10,20,30,40,50]
print(L[1:3])    # [20,30]
print(L[:3])     # [10,20,30]
print(L[2:])     # [30,40,50]
print(L[:])      # [10,20,30,40,50]
```

【文法の理解】スライスの発展パターン：刻み幅を指定すると飛び飛びに取り出せる。負のインデックスは末尾から数える（-1が最後）。刻み幅 -1 は逆順。通常の添字と異なり、範囲が末尾を越えても多くの場合エラーにならない。

```
L = [10,20,30,40,50]
print(L[::2])     # [10,30,50]
print(L[1::2])    # [20,40]
print(L[-2:])     # [40,50]
print(L[::-1])    # [50,40,30,20,10]
print(L[4:1:-2])  # [50,30]
```

【文法の理解】添字アクセスとの差：L[1] は単一の要素、L[1:2] は要素1個のリスト。L[99] は IndexError だが L[99:] は空のリストになる。step=0 は ValueError。文字列やタプルにもスライスを適用できる。

```
L = [10,20,30]
print(L[1])       # 20
print(L[1:2])     # [20]
print(L[99:])     # []
print("Python"[1:4])  # yth
```

問14 ② [1, 1, 3, 4, 5]

14. 正解：② [1, 1, 3, 4, 5] 解説：sort() はリストの要素を昇順（小さい順）に並べ替える（その場で書き換える）メソッド。[3,1,4,1,5] を昇順に並べると [1,1,3,4,5] になる。

【文法の理解】リストの複製・並べ替え・反転：リスト * n は各要素を n 倍するのではなく、並びを n 回繰り返す。sort() と reverse() は元のリストを変更し、戻り値は None。sorted() と reversed() は別の使い方。

```
L = [3,1,2]
print(L * 2)  # [3,1,2,3,1,2]
L.sort()
print(L)  # [1,2,3]
L.reverse()
print(L)  # [3,2,1]
```

【文法の理解】破壊的操作と戻り値：sort() や reverse() の戻り値を代入すると None になってしまう。元のデータを残すなら sorted(L)、逆順リストなら L[::-1] を使える。

```
a = [3,1,2]
b = sorted(a)
print(a)  # [3,1,2]
```

```
print(b)  #  [1,2,3]
```

問15 ③ while 'x' in data: data.remove('x')

15. 正解：③ while 'x' in data: data.remove('x') 解説：remove('x') は最初に見つかった 'x' を1つだけ削除するメソッドなので、1回の呼び出しでは複数ある 'x' をすべて削除 できない（①は不十分）。'x' がリスト内に存在する限り繰り返し remove する③の書き方であれば、すべての 'x' を削除できる。

【文法の理解】複数の一致値をすべて取り除く：remove は1回につき1個。存在確認しながら繰り返すか、リスト内包表記で新しいリストを作る。ループ中に同じリストから無条件削除すると要素を飛ばすことがある。

```
data  =  ["x","a","x","b"]
data  =  [v  for  v  in  data  if  v  !=  "x"]
print(data)  #  ["a","b"]
```

問16 ② 0 cat / 1 dog / 2 bird（それぞれ改行）

16. 正解：② 0 cat / 1 dog / 2 bird（それぞれ改行） 解説：enumerate(words) は「インデックス番号（0から開始）」と 「要素」のペアを順に返す。そのため i は 0,1,2、w は 'cat','dog','bird' となり、それぞれ改行して表示される。

【文法の理解】for・添字・enumerate：for x in seq は要素を順に取り出す。文字列 str もシーケンスなので s[0] は先頭文字。enumerate(seq) は (位置, 要素) を順に生成し、添字と値を同時に使える。

```
for  i,  w  in  enumerate(["cat","dog"]):
        print(i,  w)    #  0  cat  /  1  dog
for  s  in  ["Python","Java"]:
        print(s[0])  #  P,  J
```

【文法の理解】enumerate の開始番号：enumerate(seq, start=1) とすれば番号を1から振れる。タプルのアンパックで i, value に同時に受け取る。

```
for  no,  name  in  enumerate(["A","B"],  start=1):
        print(no,  name)  #  1  A  /  2  B
```

問17 ③ prices.items()

17. 正解：③ prices.items() 解説：辞書からキーと値のペアを同時に取り出すメソッドは items()。 keys() はキーのみ、values() は値のみを取り出す。

【文法の理解】辞書（dict）のキーと値：辞書は「キー:値」の対応。keys() はキー、values() は値、items() は (キー,値) の組を返すビュー。for k,v in d.items(): で同時に取り出せる。

```
d  =  {"国語":80,"数学":90}
print(list(d.keys()))
print(list(d.values()))
for  k,  v  in  d.items():
        print(k,  v)
```

【文法の理解】辞書の追加・更新：d[key] = value でキーが無ければ追加、あれば上書き。存在しないキーの d[key] は KeyError になるが d.get(key, default) は代替値を返せる。

```
d  =  {"A":1}
d["B"]  =  2
d["A"]  =  3
```

```
print(d.get("C",  0))  #  0
```

問18 ② {1, 2, 3, 4, 5}

18. 正解：② {1, 2, 3, 4, 5} 解説：「|」は集合の和集合（すべての要素を合わせたもの、重複は除く）を求める演算子。A={1,2,3}, B={3,4,5} の和集合は {1,2,3,4,5}。

【文法の理解】集合 set と集合演算：集合は重複しない要素の集まり。表示順は保証されない。| は和集合、& は積集合、- は差集合、^ は対称差集合。

```
A  =  {1,2,3};  B  =  {3,4}
print(A  |  B)  #  {1,2,3,4}
print(A  &  B)  #  {3}
print(A  -  B)  #  {1,2}
print(A  ^  B)  #  {1,2,4}
```

問19 ② del d['a']

19. 正解：② del d['a'] 解説：辞書から特定のキーを削除するには del文を使い、「del 辞書名[キー]」と書く。remove() や delete() は 辞書のメソッドとしては存在しない。

【文法の理解】辞書の検索・削除：get はキーが無い場合の初期値を指定できる。in はキーの存在判定。del d[key] は削除（無ければ KeyError）、pop(key, default) なら値を取り出しつつ安全に削除可能。

```
d  =  {"A":90,"B":80}
print(d.get("C",60))  #  60
print("A"  in  d)  #  True
print(d.pop("C",None))  #  None
```

【文法の理解】辞書の in の対象："x" in d は辞書のキーを検査し、値は検査しない。値の存在判定は "x" in d.values() を使う。

```
d  =  {"a":100}
print("a"  in  d)  #  True
print(100  in  d.values())  #  True
```

問20 ③ S.upper()

20. 正解：③ S.upper() 解説：文字列を大文字に変換するメソッドは upper()。 小文字に変換する場合は lower() を使う。

【文法の理解】組み込み関数と文字列メソッド：len(seq) は要素数・文字数、count(v) は値の出現回数。upper() は大文字化、lower() は小文字化。文字列は変更不可（イミュータブル）なので変換後の新しい文字列を得る。

```
s  =  "Banana"
print(len(s))  #  6
print(s.count("a"))  #  3
print(s.upper())  #  BANANA
```

【文法の理解】count と index / find：count は出現回数、find は最初の位置（無ければ -1）、index は最初の位置（無ければ ValueError）。文字列にもリストにも count がある。

```
s  =  "banana"
print(s.count("an"))  #  2
print(s.find("z"))  #  -1
```

```
print(s.index("a"))  #  1
```

知識科目・第2問、第3問／実技科目

以下は元の解答内容を維持した参照欄です。続く「発展文法解説」で穴埋め・読解・実技の仕組みを補います。

【知識科目】第２問（穴埋め：2問） ----------

【問題１】繁盛日（来客数が極端に多い日）の検出 【１】正解：(2) mean * 1.5
解説：「平均の1.5倍を超える」データを判定する基準は 「平均値 × 1.5」。 【２】正解：(3) x > busy_line
解説：基準（1.5倍のライン）より大きいデータを「繁盛日」として 抽出するので x > busy_line が正しい条件。

【問題２】感染者数のシミュレーション（2.5倍ずつ増加） 【１】正解：(2) daily 解説：1日目の新規感染者数は daily（3人）であり、この時点での 累計 total も daily と同じ値（3）になる。 【２】正解：(2) total + daily
解説：累計感染者数は「これまでの累計」に「その日の新規感染者数 （daily）」を加算して更新する。
【３】正解：(3) n 解説：ループを抜けた時点の n（日数）が「何日目に超えたか」の 答えになる。

【知識科目】第３問（読解：1問） ---------- 正解：② n が素数かどうかを判定する。 解説：n が2未満なら素数ではないと判定し、そうでなければ 2 から $\sqrt{n}$ までの整数で n が割り切れるかどうかを調べている。 1つでも割り切れる数（約数）が見つかれば is_prime を False にする。 $\sqrt{n}$ までしか調べないのは、もし n = a × b という約数の組があれば、 必ず a か b のどちらかが $\sqrt{n}$ 以下になるという性質を利用した、 素数判定の効率化テクニックである。

【実技科目】 ----------

【問題１】正解：if bmi >= 25: 解説：BMIが25以上のとき「肥満」と判定するので、条件式は 「bmi >= 25」。

【問題２】 【１】正解：x >= average 解説：各データ x が平均値以上かどうかを判定する条件。 【２】正解：count + 1 解説：条件に合致した（平均値以上だった）データが見つかるたびに count を1ずつ増やして、該当するデータの個数を数える。

【問題３】 【１】正解：p // size 解説：価格 p を刻み幅 size（1000円）で割った商が、そのまま価格帯の区分番号になる（0〜999円→0、1000〜1999円→1、2000〜2999円→2、…）。この問題では区分が「0円から」始まっており、区分の境界（1000円、2000円…）がちょうど size の倍数と 一致しているため、余計な調整をせずシンプルに p // size だけで 正しい区分番号が求められる。 【２】正解：dosu[idx] + 1
解説：該当する価格帯の商品数（度数）を1個分増やす。 【３】正解：i in range(5)
解説：5つの価格帯すべてについて、それぞれの商品数を順番に 表示するため、i を 0〜4 の範囲でループさせる。

発展文法解説：穴埋め・読解・実技

穴埋め 問題1：平均の1.5倍

平均 mean の1.5倍は mean*1.5。「1.5倍を超える」では境界は含まないので x>busy_line。NumPy の np.mean は平均値を返す。

```
import numpy as np
a=np.array([100,100,250])
line=np.mean(a)*1.5
print(a[a>line]) # [250]
```

穴埋め 問題2：日次と累積

daily はその日増えた人数、total は累積。total=total+daily で累積を更新する。小数へ増えるモデルでは人数を整数化する規則が必要になりうるが、元問題のコードをそのまま読解する。

```
daily=3; total=daily
daily*=2.5
```

```
total+=daily
print(total)  #  10.5
```

読解：素数判定

2以上で、2〜√n の約数が無ければ素数。range(2,int(n**0.5)+1) とし、上限を含むため +1 が必要。割り切れるかは n%i==0。見つけたら break してよい。

```
n=29
is_prime=n>=2
for  i  in  range(2,int(n**0.5)+1):
        if  n%i==0:
                is_prime=False
                break
print(is_prime)  #  True
```

実技1：BMI判定

以上と超えるの違いに注意。>=25 では25ちょうども含まれる。BMIの計算は体重kg/(身長mの2乗)で、比較条件と演算を分けて考える。

```
weight=70;  height=1.65
bmi=weight/height**2
print("肥満"  if  bmi>=25  else  "それ以外")
```

実技2：平均以上の個数

if x >= average: で条件を判定し、該当したら count += 1 とする。sum(条件式 for x in data) でも真を1、偽を0として数えられる。

```
data=[1,2,3,4]
avg=sum(data)/len(data)
count=sum(x>=avg  for  x  in  data)
print(count)  #  2
```

実技3：価格帯別度数

0〜999円は0、1000〜1999円は1となるので p//1000 を添字にできる。上限の区分は min(idx,4) にまとめ、dosu[idx]+=1 で集計する。負の価格は想定しない。

```
prices=[500,1500,2500,3500,9000]
dosu=[0]*5
for  p  in  prices:
        idx=min(p//1000,4)
        dosu[idx]+=1
print(dosu)  #  [1,1,1,1,1]
```