日商プログラミング検定 STANDARD（Python）
セット2 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 ① _

1. 正解：① _ 解説：Python の対話モードでは、直前に評価した式の結果が特殊変数「_」に自動的に保存される。ans や last、# は特殊変数ではない。

【文法の理解】対話モードの特殊変数：標準の対話モードでは、直前に評価した式の値を _ で参照できる。ただし通常のプログラムで以前の計算を保持するには変数に代入する方が明確。

```
>>> 8 * 4
32
>>> _ + 1
33
```

問2 ③ 1024

2. 正解：③ 1024 解説：「**」はべき乗演算子。2の10乗＝1024。

【文法の理解】べき乗演算子：** はべき乗。演算子 ^ はべき乗ではなく整数のビット排他的論理和。負の指数では逆数になる。

```
print(2 ** 10) # 1024
print(2 ** -2) # 0.25
print(2 ^ 3)   # 1
```

問3 ② # これはコメントです

3. 正解：② # これはコメントです 解説：Python の行コメントは「#」から行末までがコメント扱いになる。「//」はC言語系、「/* */」もC言語系のコメント記法であり Pythonでは使用できない（構文エラーになる）。

【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：「/」は実数（浮動小数点数）の商を返す演算子。「//」は切り捨て除算（商を整数に丸める）、「%」は剰余を返す。

【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 ③ 1total

5. 正解：③ 1total 解説：Python の変数名は数字から始めることができない。 1total は変数名として使用不可。total_1、_sum、Total はいずれも 使用可能（Total と total_1 は大文字小文字を区別する別の名前として 有効）。

【文法の理解】識別子の命名規則：変数名には英数字・アンダースコアなどが使えるが、数字から開始できず、予約語は不可。大文字小文字を区別する。一般には snake_case を推奨する。

```
total_1 = 10
_total = 20
Total = 30
# 1total = 10 は構文エラー
```

問6 ② A-B

6. 正解：② A-B 解説：print() の sep 引数は、複数の値を表示するときの区切り文字を 指定する。sep='-' なので、'A' と 'B' の間が「-」で区切られ 「A-B」と表示される。

【文法の理解】print()の引数：print は引数ごとに式を評価し、標準では半角空白で区切り改行する。sep= は区切り、end= は末尾の文字列。文字列の引用符は表示されない。

```
print(3 * 2, "Hi") # 6 Hi
print("A", "B", sep="-") # A-B
print("A", end="!")
```

問7 ① (３)と(４)

7. 正解：① (３)と(４) 解説： (1) not True → False (2) True and False → False (3) not False → True (4) False or True → True True 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 ③ 24 >> 2

8. 正解：③ 24 >> 2 解説：シフト演算はビットをずらす操作。 ① 2 << 1 = 4（2進数 10 を左に1桁 → 100 = 4） ② 2 << 2 = 8（10 を左に2桁 → 1000 = 8） ③ 24 >> 2 = 6（24 = 11000 を右に2桁 → 110 = 6） ④ 24 >> 3 = 3（11000 を右に3桁 → 11 = 3） 結果が6になるのは③。

【文法の理解】ビット演算と2進数：& はビットAND、| はビットOR、^ はビットXOR、~ は反転、<< と >> はシフト。bit演算と真偽の and/or は別物。

```
print(bin(5)) # 0b101
print(5 & 2) # 0
```

```
print(5 | 2)        #  7
print(24 >> 2)    #  6
```

問9 ③ C

9. 正解：③ C 解説：x=5, y=10 のとき、x > y は False、elif x == y も False（5と10は等しくない）なので、else節の print('C') 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：「x が偶数のとき『偶数』と表示、奇数なら何もしない」という 仕様に対して、③「if x % 2 == 1: print('偶数')」は x が奇数の ときに『偶数』と表示してしまい、条件が逆になっているため誤り。
①②④はいずれも「偶数のときだけ『偶数』と表示する」という 同じ動作になる。

【文法の理解】比較演算子と境界値：>= は以上（境界を含む）、> は超える（含まない）。!= は不一致、== は等しい。条件の逆転はバグの原因。境界直前・境界値・境界直後をテストする。

```
for x in [9,10,11]:
        if x >= 10: print(x, "OK")
for x in [2,3,4]:
        if x % 2 == 0: print(x, "偶数")
```

問11 ② 5（doneは表示されない）

11. 正解：② 5（doneは表示されない） 解説：while...else の else節は、ループがbreakされずに正常終了したときのみ実行される。 x=20→15(15!=5)→10(10!=5)→5(5==5、breakする！) breakが実行されたため else節（print('done')）は実行されない。 ループを抜けた後、print(x) が実行され、x=5 が表示される。

【文法の理解】while と else / break：while は条件が真の間、繰り返す。else は break せずに条件が偽になって終了したときのみ実行。break はループ終了、continue は次の反復へ進む。

```
x = 1
while x < 5:
        x *= 2
else:
        print(x)  #  8
#  breakがあればelseをスキップ
```

問12 ① 5 4 3 2 1

12. 正解：① 5 4 3 2 1 解説：range(5, 0, -1) は 5 から始まり 0 未満（0は含まない）まで -1ずつ減少するので 5,4,3,2,1 を生成する。end=' ' なので改行 せず半角スペース区切りで一行に表示される。

【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 ② [1, 3, 5]

13. 正解：② [1, 3, 5] 解説：L1[::2] は「先頭から末尾まで、2つおきに」要素を取り出すスライス。インデックス0,2,4の要素、つまり1,3,5が取り出される。

【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 [3, 2, 1]

14. 正解：② [3, 2, 1] 解説：reverse() はリストの並び順をその場（in-place）で逆転させる メソッド。戻り値は None だが、L1自体の中身が [3,2,1] に 書き換わるため、print(L1) では [3, 2, 1] が表示される。

【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 ② data.index(5)

15. 正解：② data.index(5) 解説：index(値) はリストの中から指定した値を検索し、最初に見つかった位置（インデックス）を返す。find() や search() というリストメソッドはPythonの標準リストには存在しない。

【文法の理解】要素の削除と検索：pop(i) は位置 i の要素を削除して返す。remove(v) は値 v に一致する最初の要素を削除。index(v) は最初の位置を返す。対象が無ければ remove/index は ValueError。

```
L = [5,8,5,9]
print(L.pop(0)) # 5
L.remove(5)
print(L) # [8,9]
```

問16 ③ Tue

16. 正解：③ Tue 解説：タプルもリストと同様に添字（インデックス）でアクセス できる。S[1] は2番目の要素 'Tue' を返す。

【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 ② menu['サラダ'] = 300

17. 正解：② menu['サラダ'] = 300 解説：辞書に新しい要素を追加・更新するには 「辞書名[キー] = 値」の形で代入する。add() や insert() という 辞書のメソッドは存在しない。

【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 {2, 3}

18. 正解：② {2, 3} 解説：「&」は集合の積集合（共通部分）を求める演算子。 A={1,2,3}, B={2,3,4} の共通部分は {2, 3}。

【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 False

19. 正解：② False 解説：「in」演算子は辞書に対して使うと「キーが存在するか」を 調べる。d には 'x' と 'y' というキーしかなく 'z' は存在しないので False。

【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.count('a')

20. 正解：③ S.count('a') 解説：count(値) は文字列やリストの中で、その値が出現する回数を 数える。find() や index() は最初に見つかった「位置」を返す メソッドであり、回数を数えるものではない。

【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】要注意データ（低スコア）の検出 【１】正解：(2) mean - 15
解説：「平均点より15点以上低い」データを判定する基準は 「平均値 - 15」。 【２】正解：(3) x <= lower_limit
解説：「15点以上低い」は「差が15以上（＝15点低い場合も含む）」
という意味なので、境界値ちょうど15点低いデータも要注意データに 含める必要がある。そのため、単なる x < lower_limit（15点より 低いデータのみ）ではなく、x <= lower_limit（15点以上低い
データすべて）が正しい条件になる。

【問題２】機械の生産数増加（1.2倍ずつ） 【１】正解：(3) 100
解説：1時間目の生産数は100個であり、この時点の累計 total も 100個になる。 【２】正解：(2) total + hourly
解説：累計生産数は「これまでの累計」に「今回新たに生産した数 （hourly）」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（時間数）が「何時間目に超えたか」の 答えになる。

【知識科目】第３問（読解：1問） ---------- 正解：① 文字列 s が回文（前から読んでも後ろから読んでも同じ）か どうかを判定する。 解説：s
の前半部分（range(len(s)//2)）について、前から i 番目の文字 （s[i]）と後ろから i
番目の文字（s[len(s)-1-i]）を1文字ずつ 比較している。1か所でも一致しない文字があれば is_ok を False にする
ことで、回文かどうかを判定するプログラムになっている。 'racecar' は実際に回文なので、実行結果は True になる。

【実技科目】 ----------

【問題１】正解：if temp >= 30: 解説：気温が30以上のとき「猛暑」と判定するので、条件式は 「temp >= 30」。

【問題２】正解：【１】x 【２】x 解説：現在調べている値 x が、これまでの最大値 max_val より大きければ max_val を x に更新し、逆にこれまでの最小値 min_val より小さければ min_val を x
に更新する。これをすべてのデータについて繰り返すことで、
最終的にリスト全体の最大値・最小値が求まり、その差（レンジ）が計算 できる。

【問題３】 【１】正解：(t - 1) // size 解説：通学時間 t を「1引いてから15（size）で割った商」にすると、t=1〜15分は商0（0〜15分の区分）、t=16〜30分は商1
（16〜30分の区分）というように、1から数える範囲を0始まりの 区分番号に変換できる。 【２】正解：dosu[idx] + 1 解説：該当する時間帯の人数（度数）を1人分増やす。 【３】正解：i in range(5)
解説：5つの時間帯すべてについて、それぞれの人数を順番に 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：平均より15点低い

「平均点より15点以上低い」は x <= mean-15。>= と <= を使うと境界を含む。NumPy 配列上では比較演算を全要素へ適用できる。

```
import numpy as np
a = np.array([40,60,80])
m = np.mean(a)
print(a[a <= m-15]) # [40]
```

穴埋め 問題2：倍率増加と累積

その時間の生産数 hourly と累積 total を分ける。倍率による増加は hourly *= 1.2、累積の更新は total += hourly。小数点以下の扱いは問題の条件・コードに従う。

```
hourly = 100
total = hourly
for n in range(2,4):
    hourly *= 1.2
    total += hourly
print(total) # 364.0
```

読解：回文判定

len(s)//2 で前半の文字位置だけを検査し、s[i] と s[len(s)-1-i] を比較する。末尾の添字は s[-1-i] とも書ける。応用として s == s[::-1] でも回文を判定できる。

```
s = "racecar"
print(s == s[::-1]) # True
print("python" == "python"[::-1]) # False
```

実技1：気温判定

if temp >= 30: は摂氏30度ちょうどを含む。30未満は else で扱う。浮動小数点数も整数30と比較できる。

```
for temp in [29.9,30.0,33.5]:
    print("猛暑" if temp >= 30 else "平常")
```

実技2：最大値と最小値

max_val, min_val を先頭要素で初期化し、全要素を比較して更新する。0で初期化するとデータがすべて負のとき誤るため避ける。最大値と最小値の差がレンジ（範囲）。

```
data = [23,45,12,67]
ma=mi=data[0]
for x in data:
    if x>ma: ma=x
    if x<mi: mi=x
print(ma-mi) # 55
```

実技3：通学時間の階級

1〜15分→0、16〜30分→1 の変換が (t-1)//15。正の整数時間を前提とする。t=0なら添字-1となるので、設問の階級「0〜15分」を完全に扱うには t=0 を特別に0へ割り当てる。

```
def bucket(t):
    if t <= 15: return 0
    return min((t-1)//15,4)
print([bucket(t) for t in [0,15,16,30,31,61]])
```