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

1. 正解：② python 解説：Python の対話モード（インタラクティブシェル）はターミナルで「python」（環境によっては「python3」）と入力して起動する。 .exe は拡張子であり起動コマンドではなく、py.run や pyexec という コマンドは存在しない。

【文法の理解】対話モードと実行環境：ターミナルから python または python3 で起動する。対話モードには >>> が表示され、式を入力するとその場で評価できる。スクリプトは python sample.py のようにファイル名を渡して実行する。

```
>>> 2 + 3
5
# 終了: exit()
```

問2 ② 3

2. 正解：② 3 解説：「//」は切り捨て除算（floor division）を行う演算子。 7 ÷ 2 = 3.5 だが、小数点以下を切り捨てて 3 になる。 「/」であれば 3.5（実数）になる。

【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

3. 正解：③ 解説：Python はブロックの区切りを波括弧やendではなく「インデント」（字下げ）で表す言語である。そのため同じブロック内の文は同じ深さのインデントでそろえる必要がある。そろっていないと IndentationError になる。

【文法の理解】インデントとブロック：if、for、while、def の末尾に : を置き、次の行から原則4スペース字下げする。同じブロックは深さをそろえる。タブとスペースの混在は避ける。

```
if 5 > 3:
    print("条件成立")
    print("同じブロック")
print("ブロック外")
```

問4 ② **

4. 正解：② ** 解説：べき乗（累乗）は「**」演算子で表す。例：2 ** 3 は 8。
「^」はPythonではビット単位のXOR（排他的論理和）であり、べき乗では ない点に注意。

【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 _total

5. 正解：③ _total 解説：Python の変数名は英字またはアンダースコアで始まる必要があり、
数字で始まる名前（2nd_value）は使用不可。また class は予約語
（キーワード）であるため変数名にできない。ハイフンを含む my-value
も使用できない（引き算と解釈されてしまう）。 アンダースコアで始まる _total は使用可能。

【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 ③ 6 Hi

6. 正解：③ 6 Hi 解説：print(3 * 2, 'Hi') は、まず 3*2 が計算されて 6 になり、 print
の複数引数はデフォルトで半角スペース区切りで表示される。
文字列はクォートを付けずに中身だけが表示されるため「6 Hi」となる。

【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 (1)と(2)

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

【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 ① 5 & 2

8. 正解：① 5 & 2 解説：ビット演算を2進数で考える。 5 = 0101, 2 = 0010 ① 5 & 2 = 0000 = 0（両方1のビットがない） ② 5 | 2 = 0111 = 7 ③ 5 ^ 3 = 0101 ^ 0011 = 0110 = 6 ④ ~0 = -1（ビット反転、Pythonでは ~x = -x-1） 結果が0になるのは①のみ。

【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 zero

9. 正解：③ zero 解説：x = 0 のとき、x > 0 は False、elif x < 0 も False なので、 else節の print('zero') 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：仕様は「x >= 10 のときOKと表示し、それ以外（10未満）のときは何もしない」というもの。①はそのまま 「if x >= 10: print('OK')」と書かれており仕様通り。②④はどちらも「10未満のときは何もしない（pass）、それ以外のときにOKと表示する」という書き方で、遠回りだが結果的に①と同じ動作に なる。一方、③の「if x < 10: print('OK')」は x が10未満のときに OKと表示してしまい、本来の仕様（10以上でOK）と論理が逆転して いるため誤り。

【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 ③ 8

11. 正解：③ 8 解説：while...else は、while がbreakされずに正常終了した場合に else節が実行される。x=1→2(1<5)、2→4(2<5)、4→8(4<5)、8は5未満でないためループ終了、 breakしていないのでelse節が実行され print(x) → 8 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 2 5 8（改行区切り）

12. 正解：① 2 5 8（改行区切り） 解説：range(2, 10, 3) は 2, 5, 8 を生成する（10は含まれず、2から3ずつ増加）。print(x) は毎回改行するので、2, 5, 8がそれぞれ 別の行に表示される。

【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 [20, 30]

13. 正解：② [20, 30] 解説：スライス L1[1:3] はインデックス1から2まで（3は含まない） を取り出す。L1 = [10,20,30,40] なので L1[1]=20, L1[2]=30 で 結果は [20, 30]。

【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 ① [1, 2, 3, 1, 2, 3]

14. 正解：① [1, 2, 3, 1, 2, 3] 解説：リストに対する「*」演算子は繰り返し結合を意味する。 [1,2,3] * 2 は要素を2回繰り返した [1,2,3,1,2,3] になる （数値のように各要素が2倍されるわけではない）。

【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.pop(0)

15. 正解：② data.pop(0) 解説：pop(インデックス) は指定した位置の要素を削除して返す。 pop(0) で先頭要素を削除できる。remove(0) は「値が0の要素」を 削除するメソッドであり、位置の指定ではない点に注意。

【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 ② PJ（それぞれ改行）

16. 正解：② PJ（それぞれ改行） 解説：for lang in S の各要素 lang は文字列 'Python', 'Java'。 lang[0] はその文字列の先頭1文字を取り出すので、'P' と 'J' が それぞれ別の行に表示される。

【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 ② scores.values()

17. 正解：② scores.values() 解説：辞書から値（value）だけを取り出すメソッドは values()。 キーだけなら keys()、キーと値のペアなら items()。

【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 ③ {1, 2, 3}

18. 正解：③ {1, 2, 3} 解説：set() は重複を取り除いた集合を作る。リスト[1,2,2,3,3,3] から重複を除くと {1, 2, 3} になる（集合なので順序は保証されない）。

【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 ③ 60

19. 正解：③ 60 解説：dict.get(キー, デフォルト値) は、キーが存在すればその値を、
存在しなければ指定したデフォルト値を返す。'C' は d に存在しない ため、デフォルト値の60が返る。

【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 len(data)

20. 正解：③ len(data) 解説：リストの要素数（長さ）を求める組み込み関数は len()。 length() や size() という関数・メソッドは標準では存在しない。

【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】外れ値（重いデータ）の検出 【１】正解：(2) mean + 10
解説：「平均より10以上大きいデータ」を判定する基準（しきい値）は 「平均値 + 10」。 【２】正解：(2) x >= threshold 解説：「10以上大きい」は「差が10以上（＝ちょうど10大きい場合も
含む）」という意味なので、境界値ちょうど10大きいデータも
「重いデータ」に含める必要がある。そのため、単なる x > threshold（10より大きいデータのみ）ではなく、 x >= threshold（10以上大きいデータすべて）が正しい条件になる。

【問題２】ハスの葉の倍々増加 【１】正解：(2) 1 解説：1日目の葉は1枚なので、初期値 leaves は1にする。
【２】正解：(3) leaves * 2 解説：「前日の倍」に増えるので、leaves = leaves * 2 で更新する。 【３】正解：(2) n 解説：ループを抜けた時点の日数 n が「何日目に超えたか」の答えに なる。leaves
は枚数であり日数ではないため誤り。

【知識科目】第３問（読解：1問） ---------- 正解：①
84と36の最大公約数を求める。 解説：これはユークリッドの互除法のプログラムである。 a, b = 84, 36 から開始し、b が 0 になるまで「a, b = b, a % b」を 繰り返す。この操作を繰り返すと最終的に a
に最大公約数が残る。 実際に計算すると、84%36=12 → 36%12=0 となり、最大公約数は12になる。

【実技科目】 ----------

【問題１】正解：if score >= 60: 解説：60点以上で合格、それ以外は不合格という条件分岐なので、 「score >= 60」を条件式にする。

【問題２】正解：total2 + abs(x - average) 解説：平均偏差は「各データと平均の差の絶対値」の合計を個数で割ったもの。差を絶対値にするために abs() を使い、total2 に加算していく。

【問題３】 【１】正解：age // 10 - 1 解説：年齢を10で割った商（10代なら1、20代なら2…）から1を引くと「10代=0、20代=1、30代=2…」という配列のインデックスに変換できる。 【２】正解：dosu[idx] + 1
解説：該当する年代の人数（度数）を1人分増やす（カウントアップ）。 【３】正解：i in range(5) 解説：dosu と labels の要素数（5区分）分だけ繰り返し、それぞれの 人数を表示するため、i は 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：NumPy としきい値

np.array(dataset) はリストを NumPy 配列に変換し、np.mean(np_list)
は平均を求める。「平均より10以上大きい」は mean + 10 以上、つまり >=
を使う。等号を含むかを日本語から読み取る。

```
import numpy as np
a = np.array([10,20,30])
mean = np.mean(a) # 20.0
print(a[a >= mean + 10]) # [30]
```

穴埋め 問題2：倍々増加

1日目1枚を初期値とし、2倍ずつ増加する。2日目2、3日目4、…、9日目256、10日目512で500を超える。while leaves < 500 は厳密には「500に達するまで」の条件であり、設問データでは500ちょうどに達しないため目的を達成する。一般に「500を超えるまで」を厳密に表すなら while leaves <= 500 が適切。

```
n = 1; leaves = 1
while leaves <= 500:
        n += 1
        leaves *= 2
print(n, leaves) # 10 512
```

読解：ユークリッドの互除法

a,b = b,a%b は右辺を先に計算して2変数へ同時代入する。最大公約数 gcd(a,b) は b が0ならa、それ以外は gcd(b,a%b) で求められる。

```
a,b = 84,36
while b != 0:
        a,b = b,a%b
print(a) # 12
```

実技1：合否判定

比較演算子 >= を使った if / else。Python は条件を括弧で囲む必要はない。ブロックをコロンとインデントで表す。

```
score = 72
if score >= 60:
        result = "合格"
else:
        result = "不合格"
```

実技2：平均偏差

平均偏差は各値と平均の差の絶対値を足し、データ件数で割る。abs() が負の差を正に変換する。round(x,1) は表示向けの丸めであり厳密な十進計算ではない。

```
data = [12,15,9,20,14,10,18]
avg = sum(data)/len(data)
mad = sum(abs(x-avg) for x in data)/len(data)
print(round(mad,1))
```

実技3：年代別度数分布

10代を添字0にするには age//10-1。50代以上は idx = min(idx,4) として最後の区分へまとめられる。dosu[idx] += 1 は1件カウントすること。10歳未満は想定しない点に注意。

```
ages = [15,25,35,45,55,70]
dosu = [0]*5
for age in ages:
        idx = min(age//10-1,4)
        dosu[idx] += 1
print(dosu) # [1,1,1,1,2]
```