Question 1
On considère la ligne de code `result = a + b`. Quelle affirmation est **incorrecte**?

- [X] `result` contiendra toujours soit un `int`, soit un `float`, soit un `str`.
- [ ] Si `a` et `b` ne sont pas du même type, l'opération peut générer une erreur.
- [ ] `a` et `b` ne doivent pas forcément être du même type pour que l'opération réussisse.
- [ ] L'opération `result = str(a) + str(b)` effectuera toujours une concaténation de chaînes de caractères.

Question 2
On considère l'extrait de code ci-dessous. Quelle expression sera toujours évaluée à `True`, en partant du principe que `a` et `b` sont deux variables de type `int` strictement positives?

```
q = a // b
r = a % b
```

- [X] `q * b + r == a`
- [ ] `q * b == a`
- [ ] `q * b == a + r`
- [ ] `q // r * b == a`

Question 3
On considère l'expression `s[n:m]`, avec `s` une chaîne de caractères non vide, et `n` et `m` deux `int` tels que $0 \leq$ `n` $\leq$ `m` $\leq$ `len(s)`. Quelle affirmation est **incorrecte**?

- [ ] On peut omettre `n` de cette expression si `n == 0` et écrire `s[:m]`.
- [X] On peut omettre `m` de cette expression si `m == len(s) - 1` et écrire `s[n:]`.
- [ ] L'expression retourne une chaîne de caractères de longueur `m - n`
- [ ] Si `n == m`, alors `s[n:m]` est toujours une chaîne de caractères vide.

Question 4
On suppose que `my_list` est une liste non vide de `int`. Quel extrait de code ne calculera **pas** dans `min_index` l'index du plus petit élément?

A)
```
min_index = 0
elem = 0
for elem in my_list:
    if elem < min_index:
        min_index = elem
```

B)
```
min_index = 0
for i, elem in enumerate(my_list):
    if elem < my_list[min_index]:
        min_index = i
```

C)
```
min = my_list[0]
min_index = 0
for i in range(1, len(my_list)):
    if my_list[i] < min:
        min = my_list[i]
        min_index = i
```

D)
```
min_index = 0
i = 1
while i < len(my_list):
    if my_list[i] < my_list[min_index]:
        min_index = i
    i += 1
```

- [X] Extrait A)
- [ ] Extrait B)
- [ ] Extrait C)
- [ ] Extrait D)

Question 5

On considère l'extrait de code ci-dessous. Quelle affirmation est **incorrecte**?

```
name: str = ... # une chaîne de caractères non vide
if name == "Alice" or "Alexandre":
    print("Ton nom commence par un A")
else:
    print("Je ne pense pas reconnaître ton nom")
```

- [ ] Une seule et même branche de ce code sera toujours exécutée, indépendamment de la valeur de `name`.
- [ ] La condition `name == "Alice"` donne `True` lorsque `name` vaut `"Alice"`, mais pas lorsque lorsque `name` vaut `"alice"`.
- [X] Il y a exactement deux valeurs possibles pour `name` qui feront afficher le message `Ton nom commence par un A`.
- [ ] L'expression `cond1 or cond2` donne `True` si au moins une des conditions `cond1` ou `cond2` vaut `True`.

Question 6

Qu'affiche cet extrait de code?

```
my_string = "Programming"
if "amm" in my_string:
    print("A", end="")
elif my_string.lower().startswith("pro"):
    print("B", end="")
if my_string.endswith("ation"):
    print("C", end="")
else:
    print("D", end="")
print("-")
```

- [ ] `ABD-`
- [X] `AD-`
- [ ] `BD-`
- [ ] `BC-`

Question 7

Que contient la variable `apollinaire` après exécution de cet extrait de code?

```
apollinaire = ["et", "l'unique", "cordeau", "des", "trompettes", "marines"]
poem = apollinaire
poem[0:1] = ["Et"]
```

- [X] `["Et", "l'unique", "cordeau", "des", "trompettes", "marines"]`
- [ ] `["Et", "et", "l'unique", "cordeau", "des", "trompettes", "marines"]`
- [ ] `["et", "l'unique", "cordeau", "des", "trompettes", "marines"]`
- [ ] `[["Et"], "l'unique", "cordeau", "des", "trompettes", "marines"]`

Question 8

Quelle affirmation sur cet extrait de code est **incorrecte**?

```
my_list: List[int] = ... # longue liste de valeurs omise
my_set: Set[int] = set(my_list)
```

- [X] Pour tout index `i` tel que $0 \leq$ `i` $<$ `len(my_list)`, `my_list[i]` donne la même valeur que `my_set[i]` si et seulement si `my_list` ne contient pas de doublon.
- [ ] Les deux tests d'appartenance `x in my_list` et `x in my_set` sont possibles, mais n'ont pas la même complexité temporelle exprimée avec la notation $\Theta(\cdot)$.
- [ ] L'expression `my_list.count(x)` avec `x` un élément de `my_set` peut donner une valeur supérieure à 1.
- [ ] Cette expression donnera toujours `True`: `len(my_set) <= len(my_list)`.

Question 9

On considère l'extrait de code ci-dessous. Qu'affiche-t-il lors de son exécution?

```
values = [[i * j for i in range(j, 1)] for j in range(5)]
print(values)
```

- [ ] `[[], [0], [0, 2], [0, 3, 6], [0, 4, 8, 12]]`
- [ ] `[[0], [1], [4], [9], [16]]`
- [X] `[[0], [], [], [], []]`
- [ ] `[[0, 0, 0, 0, 0], [1, 2, 3, 4], [4, 6, 8], [9, 12], [16]]`

Question 10

Laquelle de ces boucles n'affichera **pas** les index des éléments d'une `List[int]` stockée par la variable `numbers`?

A)
```
for i in range(len(numbers)):
    print(i)
```

B)
```
for i, elem in enumerate(numbers):
    print(i)
```

C)
```
for i in numbers:
    print(i)
```

D)
```
i: int = 0
while i < len(numbers):
    print(i)
    i += 1
```

- [ ] Extrait A)
- [ ] Extrait B)
- [X] Extrait C)
- [ ] Extrait D)

Question 11

On considère l'extrait de code ci-dessous. Qu'affiche-t-il lors de son exécution?

```
def run_twice(f: Callable[[int], int], x: int) -> int:
    print(f(f(x)))

def square(x: int) -> int:
    return x * x

run_twice(square(2), 4)
```

- [ ] `16`
- [ ] `4`
- [X] Une erreur
- [ ] `256`

`4` *(sur deux lignes)*

Question 12

On considère le code lacunaire ci-dessous:

```
____ condition1():
    print("A")
____ condition1() and condition2():
    print("B")
```

Si `condition1()` et `condition2()` sont les deux évaluées à `True`, quels doivent être, dans l'ordre, les mots clés manquants pour afficher à la fois `A` et `B` lors de l'exécution?

- [X] `if` — `if`
- [ ] `if` — `elif`
- [ ] `if` — `else if`
- [ ] `if` — `else`

Question 13

Que fait cet extrait de code avec les lignes du fichier `data.txt`? Dans les réponses ci-dessous, on considère comme *mot d'une ligne* une suite de caractères séparées d'une autre.

```
with open("data.txt", "r", encoding="utf-8") as file:
    for line in file.read().split("\n"):
        parts = line.split(" ")
        if len(parts) > 2:
            parts = parts[::-1]
        print(" ".join(parts))
```

- [X] Pour chaque ligne, afficher les mots dans l'ordre inverse s'il y a plus de deux mots et dans l'ordre du fichier sinon.
- [ ] Pour chaque ligne qui a plus de deux mots, afficher les mots dans l'ordre inverse.
- [ ] Pour chaque ligne, afficher le dernier mot s'il y a plus de deux mots sur la ligne.
- [ ] Pour chaque ligne, afficher les deux premiers mots dans l'ordre inverse.