School of Computer Science
Introduction to Programming Revision Questions

Sample

Instructions:

Confidential and closed book.
Write in Blue or Black pen only.
All code answers using Python3 language.
Non-programmable calculators permitted.
Student card must be visible during quiz.

Student Identification (SID):

.....

DO NOT WRITE ANY NAME.
Quiz only marked with SID.

Question 1

For each of the given snippets of Python3 code, what is the difference between **the last two lines of each code snippet below?** If applicable, explain the scenarios where you would use each line. If not applicable, explain why you would not use it.

(a) Snippet 1:

```
1 x = "5"
2 x = int(x)
3 int(x)
```

(b) Snippet 2:

```
1 "asdf".upper
2 "asdf".upper()
```

(c) Snippet 3:

```
1 a = ()
2 a = []
```

(d) Snippet 4:

```
1 b = 17
2 c = 5
3 a = b / c
4 a = b // c
```

(e) Snippet 5:

```
ls = []
ls.append("123")
ls = ls.append("123")
```

Question 2

Consider the following Python3 code:

```
1  def special_func_num(x, y):
2      x = 5
3      y = 10
4      return x + y
5
6  def special_func_list(x, y):
7      x[0] = 5
8      y[0] = 10
9      return x[0] + y[0]
10
11 # Part 1
12 x = 1
13 y = 2
14 print(special_func_num(x, y))
15 print(x)
16 print(y)
17
18 # Part 2
19 x = [1]
20 y = [2]
21 print(special_func_list_(x, y))
22 print(x[0])
23 print(y[0])
```

(a) What will the program print?

(b) In the line `print(special_func_num(x, y))`, what are inputs to a function in this context called? (i.e. what are `x` and `y` called?)

(c) In the line `def special_func_num(x, y)`, what are inputs to a function in this context called? (i.e. what are `x` and `y` called?)

(d) Explain what happens to `x` and `y` in the calls to the functions `special_func_num` and `special_func_list`. Include what kind of pass by operation is used. Pass by _____?

Question 3 - Flow Charts

(a) Draw a flowchart for the following block of code:

```
mark = 85
if mark >= 85:
    print("Well done!")
if mark >= 75:
    print("Nice!")
if mark >= 65:
    print("Not bad...!")
if mark >= 50:
    print("Close one...")
if mark >= 0:
    print("Well then...")
```

(b) Draw a flowchart for the following block of code:

```
mark = 85
if mark >= 85:
    print("Well done!")
elif mark >= 75:
    print("Nice!")
elif mark >= 65:
    print("Not bad...!")
elif mark >= 50:
    print("Close one...")
elif mark >= 0:
    print("Well then...")
```

(c) Which piece of code likely has the desired output?

Question 4 - Classes

(a) What is a Class? What is an Object? Describe this in words.

(b) What is a **constructor** in the context of a class? Describe this in words.

(c) What are **attributes** in the context of a class? Describe this in words.

Code Block A

```
class INFO1110Student:
    def __init__(self, name, average_mark, attendance):
        self.name = name
        # Note: do not worry about not checking for in between 0 and
            100 for the constructor.
        self.average_mark = average_mark # 0-100 (a percentage)
        self.attendance = attendance     # >0 (no. of attended labs)
        self.tutor_rating = 100       # Starts at 100. 0-100 (a
            percentage)
        self.confidence = 0           # 0-100 (a percentage)
        self.hours_studied = 0        # >0 (number of hours)

    # average_mark * tutor_rating (your connections!)
        def get_chance_of_getting_into_google(self):
              # Note: Divide by 100 to still maintain percentage
                 output
        self.average_mark * self.tutor_rating / 100

    # Satisfaction is average_mark * hours_studied
    def get_INFO1110_satisfaction(self):
        return self.average_mark * self.hours_studied

    # Generic study. Increases your average_mark by 5% per hour
    def study(hours):
        self.hours_studied += hours
        self.average_mark += 5 * hours
        # In case it goes above 100
        if self.average_mark > 100:
            self.average_mark = 100

    # Attend your lab! Increases confidence by 1% and is capped at
        100.
    def attend_lab(self):
        attendance += 1
        if self.confidence <= 100:
            self.confidence += 1

    # Crying halves your confidence and reduces your tutors rating by
         10%
    def cry(self):
        self.confidence /= 2
        self.tutor_rating - 10

me = INFO1110Student("Victor", 75, 5)
me.study(1)        # Trying to do Assignment 2
me.cry()           # Cannot do it :(
print(me.get_chance_of_getting_into_google())

me.attend_lab()    # Week 13 desperation!
me.cry()           # The lab went horribly :'(
print(me.get_INFO1110_satisfaction())
```

(d) Consider **Code Block A** on the previous page.

(i) There are 5 lines **within the class** (excluding the constructor) that each contain one small error. Locate all 5 and write the correct code. Indicate which line you are correcting for each line.

(ii) What is the final printed output of the program?

Question 5

(a) Write a Python3 **function** called `special_number(n)` that receives a parameter `n` and returns a list that contains numbers in the pattern: 2 odds, 2 evens, 2 odds, 2 evens... etc.

Example 1
An input of `4` should return a list of 4 numbers: `[1, 3, 4, 6]`

Example 2
An input of `7` should return a list of 7 numbers: `[1, 3, 4, 6, 7, 9, 10]`

Example 3
An input of `10` should return a list of 10 numbers: `[1, 3, 4, 6, 7, 9, 10, 12, 13, 15]`

Restrictions:
Only Allowed: `while` loops, `if` statements, function `len()`, `type()`, `isinstance()`, list method `append()`, string method `split()`, `format()`. keywords `elif`, `else`, `return`, `break`, `continue`, `def`, `self`, `None`, `try`, `raise`, `except`, `is`, `import sys`, and any arithmetic or boolean comparison operators.
Remember: Do not use a for loop, do not use the `in` keyword, do not use `enumerate`, do not use slices or slicing e.g. `[1:]`, do not use lambda functions e.g. `sorted()`. Marks will be deducted.

(b) Write a Python3 program `reader.py` that opens a file whose path is provided as a command line argument, and prints out the line numbers that correspond to the numbers returned from part a). Line numbers begin at `1`.

Example

`some_file.txt` contains:

```
Hey!
We hope you enjoyed
INFO1110!
Please continue
to study well
and have fun
exploring more code
```

`python3 reader.py some_file.txt` outputs:

```
Hey!
INFO1110!
Please continue
and have fun
exploring more code
```

Restrictions:

Only Allowed: `while` loops, `if` statements, function `len()`, `type()`, `isinstance()`, list method `append()`, string method `split()`, `format()`. keywords `elif`, `else`, `return`, `break`, `continue`, `def`, `self`, `None`, `try`, `raise`, `except`, `is`, `import sys`, and any arithmetic or boolean comparison operators.

Remember: Do not use a for loop, do not use the `in` keyword, do not use `enumerate`, do not use slices or slicing e.g. `[1:]`, do not use lambda functions e.g. `sorted()`. Marks will be deducted.

Write your answer on the following page.

Write your answer to **Question 5.b)** here.

Question 6

(a) Write a Python3 **function** called `modulus(num, divisor)` that returns the remainder of `num` when divided by `divisor`. That is, write your own modulus function.

For example, the remainder of `17` when divided by `5` is `2`. The remainder of `4` when divided by `3` is 1.

Note that for the first example, the value of `num` is `17`, and the value of `divisor` is `5`. If either input passed to the function are not integers, raise a `TypeError` with the message: `"Input(s) are not integers."`

You **may not** use the `%` (modulus) operator, or the `/` or `//` operators. (i.e. No division operators can be used in the construction of your modulus function.)

Examples

```
print(modulus(16, 3))       # Should display 1.
print(modulus(-2, 3))       # Should display 1.
print(modulus(-16, 7))      # Should display 5.
print(modulus(True, 1))     # Should raise a TypeError.
```

Restrictions:
Only Allowed: `while` loops, `if` statements, function `len()`, `type()`, `isinstance()`, list method `append()`, string method `split()`, `format()`. keywords `elif`, `else`, `return`, `break`, `continue`, `def`, `self`, `None`, `try`, `raise`, `except`, `is`, `import sys`, and any arithmetic or boolean comparison operators.
Remember: Do not use a for loop, do not use the `in` keyword, do not use `enumerate`, do not use slices or slicing e.g. `[1:]`, do not use lambda functions e.g. `sorted()`. Marks will be deducted.

(b) Write a Python3 program `is_prime.py` that reads in an integer as a command line argument. It will print `"Prime"` or `"Not prime"`, depending on whether the number received is a prime number or not.

If the provided command line argument cannot be converted to an integer, print `"Invalid input"`.

Hint: A prime number is a number that cannot be divided by any number except 1 and itself. Use your `modulus` function from **Question 6.a)** to see if there are any numbers that can divide into the given number.

Examples

```
python3 is_prime.py 5
Prime

python3 is_prime.py 10
Not prime

python3 is_prime.py Prime
Invalid input
```

Restrictions:
Only Allowed: `while` loops, `if` statements, function `len()`, `type()`, `isinstance()`, list method `append()`, string method `split()`, `format()`. keywords `elif`, `else`, `return`, `break`, `continue`, `def`, `self`, `None`, `try`, `raise`, `except`, `is`, `import sys`, and any arithmetic or boolean comparison operators.
Remember: Do not use a for loop, do not use the `in` keyword, do not use `enumerate`, do not use slices or slicing e.g. `[1:]`, do not use lambda functions e.g. `sorted()`. Marks will be deducted.

Write your answer on the following page.

Write your answer to **Question 6.b)** here.

Notes and reminders

Common types: `int`, `float`, `bool`, `str`

Common escape characters: `\n \t \' \" \\`

List terms: `[ ]`

Type information: `isinstance(object, class)`, type(object)

Parsing methods: `int()`, `float()`, `str()`

Collection methods and functions: `append`, `insert`, `remove`, `pop`, `index`, `reverse`, `copy`

Keywords

`def` – Keyword for defining a function

`import` – Imports a module for your program

`raise` – Raises an exception

`break`, `continue`, `return`, `assert`

Common Modules

`sys` – module, provides access to command line arguments.

`iterable` **collection operations:**

`list(iterable)` – returns a copy of list whose items are the same and in the same order as iterable items

`len()` – returns the length of the iterable collection

Control Flow Syntax:

- try: ... except [... as ...]: ... [finally: ...]
- while ... : ... [else: ...]
- if ... : ... [elif ... : ...] [else: ...]

Common Exceptions: `Exception`, `EOFError`, `ZeroDivisionError`, `SyntaxError`, `AssertionError`, `KeyError`, `FileNotFoundError`, `IndexError`, `NameError`.

Arithmetic Operators: `+ - * / // **`

Logical Operators: `not and or == != >= <= < >`

Assignment Operators: `= += *= -= /=`

Useful functions and methods

`input()` – Allows for reading from standard input

String methods (`str`):

`==` – Check if this string is equal to the other string

`s[i]` – Gets a character from `s` at index `i`

`lower` – Returns a string in lowercase

`upper` – Returns a string in uppercase

`strip` – Returns a string with whitespace stripped from start and end of the string (pattern matched).

`format` – Returns a formatted version of the string. Using substitution from mapping. Substitutions are identified by braces { and }