SAMPLE QUESTIONS FOR COMP110029.01

PAN

[Note: "*" means the easiest and "*****" means the hardest. Rule followed.]

1. Variable & Type

(1) Consider the code below [**]

```
x = 1
y = 2 * x
z = 3 ** y
h = 4 * z
print(h)
```

.

What is the output of this program?

a) 24 b) 15 c) 0 d) 36

(2) Consider the code below [**]

```
x = 1
y = x / 5
z = x / 5 + 1.0
```

.

(a) Variables y, z are of type

a) Integer, Float b) Integer, Integer c) Float, Float d) Float, Integer

(b) Variables y, z are of value

a) 0, 1.0 b) 0.2, 1.2 c) 0.2, 1.0 d) 0, 1.2

(3) Consider the code below [***]

```
1 x = "hello"
  y = " world"
3 n = 7
  d= x+y
5 print(x+n)
```

.

What is the output of this program?

a) hello7 b) hello 7 c) Error at line 4 d) Error at line 5

2. Conditional

(1) Consider the code below [***]

```python
x = 10
out = 0
if x % 2 == 0:
    out = out + 1
else:
    out = out + 2

if x % 2 ** 3 == 2:
    out = out + 1
else:
    out = out + 2
print(out)
```

.

What is the output of this program?

a) 2 b) 3 c) 4 d) 6

(2) Consider the code below [***]

```python
1  out = "I like "
2  x = 24
3  if x % 2 != 0:
4      x = x / 2
5  x = x / 2
6  if x / 5 == 1:
7      out = out + "throwing "
8
9  if x % 4 == 0:
10     out = out + "tomato"
11 else:
12     out = out + "potato"
13
14 print(out)
15 print(x)
```

.

(a) What's the output of line **14**?

a) I like throwing tomato b) I like throwing potato

c) I like tomato d) I like potato

(b) What's the output of line **15**?

a) 24 b) 12 c) 6 d) 6.0

3. Loop & List

(1) Consider the code below [**]

```
a = 1
b = 2
while (a < 11):
  a = a + b
print(a)
```

.

What is the output of this program?

a) 9   b) 11   c) 13   d) infinite loop

(2) Consider the code below [****]

```
a = 1
b = 2
c = 0
while (c  < b):
  c = a - b
  a = c + b
  b = c
print(a)
```

.

What's the output of this program?

a) 0   b) infinite loop   c) -1   d) 1

(3) Consider the code below [**]

```
a = [1,2,3,4]
out = 0
w = 1
for i in a:
  out = out + i * w
  w = w * 2
```

.

Variables *out*, w are of value

a) 98, 16   b) 98, 8   c) 49, 16   d) 10, 1

(4) Consider the code below [****]

```
  a = [0,1,2,3,4,5]
2 b = list()
  for i in a[1:]:
4   b.append(a[i-1] + 2 * a[i]);
  print(b)
```

.

What's the output of this program?

a) $[2, 5, 8, 11, 14]$ b) Error at line 4

c) Error at line 3   d) $[2, 5, 8, 11, 15]$

(5) Consider the code below [***]

```
  a = "hello morikA"
2 b = ""
  for i in a:
4   if i in "aeiou":
      b = b + "*";
6   else:
      b = b + i.upper();
8 print(b)
```

.

What's the output of this program?

a) H*LL* M*R*k*

b) H*LL* M*R*kA

c) Error at line 4

d) HELLO MORIk*

4. Dictionary

(1) Consider the code below [**]

```
  pan = {
2   "idea" : 4,
    "chalk" : 1,
4   "notebook" : 3
  }
6 pan["chalk"] -= 1;
  print(pan["chalk"] + pan.get("proof", 1))
8 pan["proof"] = 2;
  print(pan.get("proof", 1) * pan["idea"] - 2*pan["notebook"] )
```

.

What are the outputs respectively of line 7 & 9?

a) 1, -2

b) 2, -2

c) 1, 2

d) Error at line 7

5. Function

(1) Consider the code below [***]

```
def my_max(x, y):
  if x > y:
    return 2*x
  else:
    return y

def my_min(x, y):
  if x < y:
    return x / 2
  return y

mys = my_min(10, my_max(3, 1));
print(mys);
```

.

What is the output of the program?

a) 6 b) 3 c) 2 d) 5

(2) Consider the code below [***]

```
def factorial(n):
  out = 1
  for i in range(0, n):
    out = out * i
  return out

print(factorial(5))
```

.

What is the output of the program?

a) 120 b) 24 c) 0 d) infinite loop

6. Integrated Skills

(1) Given a piece of code below (***)

```
def greet("P")
  print (P + "is a good guy!")
  return P + "is a bad guy .."

P = "Patrick-"+ 11 + " ";
print(greet(P))
```

.

(a) find out the potential syntax errors and make it runnable [Tips: Do it directly beside the original code]

(b) What is the output of your modified code?

(2) Coding according to the description below (***),

"

By 1950, the word *algorithm* was most frequently associated with "*Euclid's algorithm*", a process for finding the greatest common divisor of two numbers which appears in Euclid's *Elements* (book vii, propositions 1 and ii). It will be instructive to exhibit Euclid's algorithm here:

Algorithm E(*Euclid's algorithm*). Given two positive integers m and n, find their greatest common divisor, i.e., the largest positive integer which evenly divides both m and n.

E1. [Find remainder.] Divide m by n and let r be the remainder. (we will have $0 \leq r < n$)

E2. [Is it zero?] If r = 0, the algorithm terminates; n is the answer.

E3. [Interchange.] Set $m \leftarrow n$, $n \leftarrow r$, and go back to step E1. |

"

– from *D.Knuth, The Art of Computer Programming, 2nd, Vol. 1, Page 2*

```
def gcd(m, n):
  ## YOUR CODES. RETURN THE GREATEST COMMON DIVISOR
  ## OF GIVEN INPUTS AS POSITIVE INTEGERS m, n
```

.

7. Solutions with Explanations

1 (1) **d)** $x = 1 \rightarrow y = 2 \times 1 \rightarrow z = 3^2 \rightarrow h = 4 \times 9$

(2.a) **a)** line 2: $y = x/5$; Since both x and 5 are integers, so as y. Although (line 3) $x/5$ is integer, 1.0 is float. Thus z is float.

(2.b) **a)** According to the precedence table of arithmetic operator, $/ > +$. Thus $x/5$ is first evaluated, which gives value of 0 (Integer, also the value for y). Next, $0 + 1.0$, which yields 1.0, assigned to z in line 3

(3) **d)** $x + y$ is OK since they are both of type string, which lets the addition behave as concatenation. However, in line 5, n is of type integer (from line 3). The behavior of addition between string and integer is undefined in Python.

2 (1) **a)** in the first conditional, $x\%2 = 10\%2 == 0$ is satisfied, which leads to the execution of $out = out + 1$ (line 4). The next conditional, $x\%2 * *3 = x\%8 = 10\%8 == 2$ is satisfied (the precedence rule, $** > \%$), which leads to the execution of $out = out + 1$ (line 9). Thus out is of value 2.

(2.a) **c)** in the first conditional, $x\%2 = 24\%2 = 0! = 0$ is **not** satisfied. Thus line 4 $x = x/2$ is **not** executed. Line 5 is unconditionally executed by noticing there is no indentation at the start of the line. Thus $x \leftarrow 24/2 = 12$. The conditional on line 6, $x/5 = 12/5 = 2 == 1$ is again **not** satisfied. Thus line 7 won't be executed. For the conditional on line 9, $x\%4 = 12\%4 = 0 == 0$ is satisfied, which updates out $\leftarrow$ "I like " + "*tomato*". Thus line 14 will print "I like tomato".

(2.b) **b)** Only line 5 updated x, which means x is of value 12 eventually.

3 (1) **b)** Since b is always of value 2, a is incremented by b (i.e. 2) during each iteration. a takes the value by sequence $1, 3, \ldots, 9, 11, 13$. While-loop will be break-ed the first time when the condition is unsatisfied, which tells us the final value of a in line 5 should be 11, $11 < 13$

(2) **d)** At first we have $(a = 1, b = 2, c = 0)$. The $c < b$ is satisfied, which lets the program goes into the iteration part for the first time. Line 5 indicates $c \leftarrow a - b = 1 - 2 = -1$. Line 6 updates a with $a \leftarrow c + b = (\mathbf{-1}) + 2 = 1$. Line 7, however, simply lets $b \leftarrow c = \mathbf{-1}$, which simultaneously means the ending of the first iteration. Check the condition $c < b \rightarrow (-1) < (-1)$ is unsatisfied thus the loop is break-ed. The program comes to line 8, when a is of value 1.

(3) **c)** The **for-in** loop behaves as a look-through of the list a. During each iteration,
$out \leftarrow 0 + 1 \times 1 = 1$, $w \leftarrow 1 \times 2 = 2$.
$out \leftarrow 1 + 2 \times 2 = 5$, $w \leftarrow 2 \times 2 = 4$.

$out \leftarrow 5 + 3 \times 4 = 17$, $w \leftarrow 4 \times 2 = 8$
$out \leftarrow 17 + 4 \times 8 = 49$, $w \leftarrow 8 \times 2 = 16$

(4) **a)** At first, a[1:] in line 3 means a slice of a from the index-1 element to the end, which gives a new list $[1, 2, 3, 4, 5]$. Then the loop is no more than a look-through of **the** new list, with i takes the value of corresponding element subsequently. During each iteration,
b.append(a[0] + 2 * a[1]) ≡ b.append(0 + 2 * 1) ≡ b.append(2)
b.append(a[1] + 2 * a[2]) ≡ b.append(1 + 2 * 2) ≡ b.append(5)
b.append(a[2] + 2 * a[3]) ≡ b.append(2 + 2 * 3) ≡ b.append(8)
b.append(a[3] + 2 * a[4]) ≡ b.append(3 + 2 * 4) ≡ b.append(11)
b.append(a[4] + 2 * a[5]) ≡ b.append(4 + 2 * 5) ≡ b.append(14)

(5) **b)** Consider a string as a list of characters (i.e. letters). At the beginning of each iteration, i takes the value of each character sequentially. With line 4 checks whether the current i is one of "aeiou" (the vowel **and certainly case-sensitive**.), the lower case vowel character in the string a will be replaced with a steroid $*$ in the transformed string b, while the line 6-7 **else** replaces other character in the string a with their upper-case counterpart in the target b. Thus b is **H*LL* M*R*kA**

4 (1) **c)** The behavior of *pan.get("proof", 1)* is as, if there is no key *"proof"* in the dictionary *pan* (line 7), it returns the default value 1 instead of crashing the program awkwardly. Otherwise (line 9) simply do the same thing as *pan["proof"]*. Thus (be careful of the updating of value for key *"chalk"* in line 6, which means the same as *pan["chalk"] = pan["chalk"] - 1*)
Line 7: pan["chalk"] + pan.get("proof", 1) = **0** + 1 = **1**
Line 9: pan.get("proof", 1) * pan["idea"] - 2 * pan["notebook"] $= 2 \times 4 - 2 \times 3 = \mathbf{2}$

5 (1) **a)** First invoke the function *my_max* with input $(x = 3, y = 1)$. Go to line 2 in its definition, where the conditional $x > y$ is satisfied, which leads to **return 2*x** $= 2 \times (3) = \mathbf{6}$. Now line 12 becomes *mys = my_min(10, 6)*. Invoke *my_min* with input $(x = 10, y = 6)$. Go to line 8 in its definition, $x < y \equiv 10 < 6$ is **not** satisfied, which leads the program to line 10, **return y** = **6**. Thus line 12 becomes *mys = 6*.

(2) **c)** line 7 invokes the function *factorial* with input $(n = 5)$. Honestly execute the codes in definition by hand instead of envisioning from the name itself. As in line 3 in definition, the *range(0, n)≡range(0,5)* simply means generate a list as $[0, 1, 2, 3, 4]$. Notice in the first iteration, line 4 indicates $out \leftarrow 1 \times i = 1 \times 0 = 0$. Thus no matter how the loop continues (under the condition of finiteness), the return-ed value will always be 0. Thus line 7 becomes **print(0)**

6 (1) (a) [A possible version]

```
def greet(P):
  print (P + "is a good guy!")
  return P + "is a bad guy .."
```

```
P = "Patrick-"+str(11) + " ";
print(greet(P))
```

.

(b) Output corresponds to the modified version above
Patrick-11 is a good guy!
Patrick-11 is a bad guy ..

(2) [A possible implementation]

```
def gcd(m, n):
  ## YOUR CODES. RETURN THE GREATEST COMMON DIVISOR
  ## OF GIVEN INPUTS AS POSITIVE INTEGERS m, n
  r = m - (m / n) * n   # Divide m by n and let r be the remainder.
  while(r > 0):
    m = n    # Set m <- n, n <- r
    n = r
    r = m - (m / n) * n  # go back to step E1
  return n       # If r = 0, n is the answer.
```

.