CSCI 145 Exam 4 Practice — Lectures 15–20 ANSWER KEY

50 minutes. You may use the Exam 4 reference sheet; no other notes, devices, computers, or calculators.
Answer as many questions as you can; if one stalls you, move on and come back to it later if time allows.
When asked for a value, write it as Python would show it (for example, `3.0` vs. `3`, `'hi'` for a string, `(2, 3)` for a tuple).

PART 1 — BITS, BINARY, AND HEX (LECTURE 15)

1. Convert each number. The binary numbers are unsigned. Use the hex table on the reference sheet if needed.

`100101` in decimal	**`37`**	70 in binary	**`1000110`**
9C (hex) in decimal	**`156`**	200 in hex	**`C8`**
`11100101` in hex	**`E5`**	6B (hex) in binary	**`01101011`**

2. Answer each question. All the numbers are unsigned: 0 and up.

a) How many different patterns can 10 bits hold? **`1024`**

b) What is the largest unsigned whole number 10 bits can hold? **`1023`**

c) What is the fewest bits that can hold every unsigned whole number from 0 to 40? **`6`**

d) What is `1101` + `0110`, in binary? **`10011`**

3. Colors and Python.

a) Write `#4080C0` as a (red, green, blue) tuple. **`(64, 128, 192)`**

b) Write (0, 255, 170) as a hex color code. **`#00FFAA`**

c) Write the value Python gives for each expression. Put quotes around a string.

`bin(9)`	**`'0b1001'`**	`hex(31)`	**`'0x1f'`**
`int("A2", 16)`	**`162`**	`0x11 + 1`	**`18`**

PART 2 — NEGATIVE NUMBERS, FLOATS, AND TEXT (LECTURE 16)

4. Use signed 8-bit two's complement.

a) `11110101` in decimal **`-11`**

b) `00110111` in decimal **`55`**

d) What are the smallest and largest numbers that signed **6-bit** two's complement can hold?

smallest **`-32`** largest **`31`**

c) Write −42 in signed 8-bit two's complement. Show your work.

```
 42 is   00101010
flip:    11010101
add 1: +        1
-42 is   11010110
```

5. Floats.

Write True or False for each.

a) `0.5 + 0.25 == 0.75` — **True**

b) `0.3 + 0.6 == 0.9` — **False**

c) A Python int is exact, however big it is. — **True**

d) Between 0.1 and 0.2 there are numbers that a float can't store exactly. — **True**

e) a and b are floats. Write an expression that is `True` when they are less than 0.001 apart. — **abs(a - b) < 0.001**

6. Write the value Python gives for each expression. Put quotes around a string. Use the ASCII table on the reference sheet.

`ord("R")` — **82**

`chr(ord("E") + 32)` — **'e'**

`"apple" < "Zebra"` — **False**

`chr(0x51)` — **'Q'**

These hex bytes are ASCII codes. What word do they spell? `50 69 78 65 6C` — **Pixel**

PART 3 — IMAGES (LECTURE 17)

7. A grayscale image `pic` is 7 pixels wide and 5 pixels tall.

a) How many pixels does it have? — **35**

b) What expression gives its top-right pixel? — **pic[6][0]**

c) What expression gives its bottom-left pixel? — **pic[0][4]**

8. These lines run in the Shell, in order. Then write the value Python gives for each expression. If an expression causes an error, write error.

```
>>> img = middimage.new(5, 2, "L")
>>> img[4][1] = 180
>>> img[4][1] = img[4][1] + 100
>>> img[1][0] = img[4][1] // 2
```

`img[4][1]` — **255**

`img[1][0]` — **127**

`img[0][1]` — **0**

`img[5][0]` — **error**

9. This code makes a new grayscale image and sets every pixel. Write each pixel's value in the grid.

```
img = middimage.new(4, 2, "L")
for x in range(img.width):
    for y in range(img.height):
        img[x][y] = 60 * (x + y)
```

	x = 0	x = 1	x = 2	x = 3
y = 0	**0**	**60**	**120**	**180**
y = 1	**60**	**120**	**180**	**240**

10. Write a function `dim_top(img)`. `img` is a grayscale image. The function returns a new grayscale image of the same size. In the new image, each pixel in the top half (where `y` is less than `img.height // 2`) has half the value of the same pixel in `img`, as an int. Every other pixel has the same value as in `img`. The function does not change `img`. Do not include a docstring. For example:

`tiny`

	0	1	2
0	250	99	40
1	100	7	180

`dim_top(tiny)` returns

	0	1	2
0	125	49	20
1	100	7	180

```
def dim_top(img):
    result = middimage.new(img.width, img.height, "L")
    for x in range(img.width):
        for y in range(img.height):
            if y < img.height // 2:
                result[x][y] = int(img[x][y] / 2)
            else:
                result[x][y] = img[x][y]
    return result
```

PART 4 — OBJECTS (LECTURE 18)

11. Consider the class `Battery` and the code to its right.

```
class Battery:
    def __init__(self, name, level):
        self.name = name
        self.level = level

    def __str__(self):
        return self.name + ": " + str(self.level)

    def drain(self, amount):
        self.level = self.level - amount

    def fuller_than(self, other):
        return self.level > other.level
```

```
p = Battery("Phone", 80)
q = Battery("Watch", 50)
r = q
r.drain(30)
p.drain(10)
q.name = "Band"
```

a) After this code runs, write the value of each expression.

`p.level` **70**

`q.level` **20**

`r.name` **'Band'**

`p.fuller_than(q)` **True**

`str(r)` **'Band: 20'**

b) How many `Battery` objects does the code make? **2**

c) What happens when the line `b = Battery("Lamp")` runs? **an error (TypeError)**

12. Write a class `Cart` with these four methods. Do not include docstrings.

- `__init__(self, owner)` stores `owner` as the attribute `owner`, and sets the attribute `prices` to an empty list.
- `add(self, price)` adds `price` to the end of the list `prices`.
- `total(self)` returns the sum of the prices. If the cart is empty, it returns `0`.
- `__str__(self)` returns a string such as `"Ana: 2 items"`.

For example:

```
>>> cart = Cart("Ana")
>>> cart.total()
0
>>> cart.add(4)
>>> cart.add(9)
>>> cart.total()
13
>>> print(cart)
Ana: 2 items
```

```
class Cart:
    def __init__(self, owner):
        self.owner = owner
        self.prices = []

    def add(self, price):
        self.prices.append(price)

    def total(self):
        total = 0
        for price in self.prices:
            total = total + price
        return total

    def __str__(self):
        count = str(len(self.prices))
        return self.owner + ": " + count + " items"
```

PART 5 — SPECIAL METHODS, INHERITANCE, AND POLYMORPHISM (LECTURE 19)

13. The class `Duration` is below. The three lines on the right run in the Shell. Then write what the Shell shows for each line under them.

```
class Duration:
    def __init__(self, mins):
        self.mins = mins
    def __str__(self):
        return str(self.mins) + " min"
    def __lt__(self, other):
        return self.mins < other.mins
    def __add__(self, other):
        return Duration(self.mins + other.mins)
```

```
>>> a = Duration(45)
>>> b = Duration(45)
>>> c = a
```

```
>>> a == b          False
>>> a == c          True
>>> a < b + c       True
>>> print(a + b)    90 min
```

14. Consider the classes `Ticket`, `StudentTicket`, and `VipTicket`, and the code to their right.

```
class Ticket:
    def __init__(self, event, price):
        self.event = event
        self.price = price
    def cost(self):
        return self.price
    def label(self):
        return "regular"
    def __str__(self):
        text = self.label() + " ticket: $"
        return text + str(self.cost())

class StudentTicket(Ticket):
    def cost(self):
        return self.price // 2
    def label(self):
        return "student"

class VipTicket(Ticket):
    def __init__(self, event, price, seat):
        super().__init__(event, price)
        self.seat = seat
    def cost(self):
        return self.price + 20
```

```
t = Ticket("Play", 30)
s = StudentTicket("Play", 30)
v = VipTicket("Play", 50, "A1")
for ticket in [t, s, v]:
    print(ticket)
print(v.seat, v.label())
print(s.cost() + v.cost())
```

a) What does the code print?

```
regular ticket: $30
student ticket: $15
regular ticket: $70
A1 regular
85
```

b) Whose `__init__` runs when `s` is made: `Ticket`'s, `StudentTicket`'s, or `VipTicket`'s?

Ticket's

c) Suppose `VipTicket`'s `__init__` did not have the line `super().__init__(event, price)`. What would happen when the loop prints `v`?

an error (AttributeError)

15. Write a class `GroupTicket`, a child class of `Ticket` from question 14. Do not include docstrings.

- `GroupTicket("Play", 30, 4)` makes a ticket with the event `"Play"`, the price `30`, and a new attribute `size` that is 4. Use `super().__init__`.
- Its `cost` method returns the price times the size.
- Its `label` method returns `"group"`.

For example:

```
>>> g = GroupTicket("Play", 30, 4)
>>> g.size
4
>>> print(g)
group ticket: $120
```

```
class GroupTicket(Ticket):
    def __init__(self, event, price, size):
        super().__init__(event, price)
        self.size = size

    def cost(self):
        return self.price * self.size

    def label(self):
        return "group"
```

PART 6 — ANIMATION AND GAMES (LECTURE 20)

In this part, the screen is 600 pixels wide and 400 pixels tall: `SCREEN_WIDTH` is 600 and `SCREEN_HEIGHT` is 400. The classes `Ball`, `Player`, and `Rock` are the ones from the videos, as on the reference sheet.

16. The game loop on the reference sheet shows one moving ball. Each change below is made to it, one at a time. Write the letter (A, B, C, or D) of what happens.

a) The line `screen.fill((255, 255, 255))` is removed **C**

b) The line `pygame.display.flip()` is removed **A**

c) The line `clock.tick(60)` is removed **D**

d) `running = False` is changed to `running = True` **B**

A. The window never shows the ball
B. Closing the window doesn't end the loop
C. The ball leaves a trail behind it
D. The ball moves much too fast

17. These two lines draw a rectangle and then a circle.

```
pygame.draw.rect(screen, (255, 0, 0), (40, 300, 120, 60))
pygame.draw.circle(screen, (0, 0, 255), (160, 300), 50)
```

a) What color is the rectangle? **red**

b) Where is the rectangle's lower-right corner? **(160, 360)**

c) What is the x of the left edge of the circle? **110**

d) Where the two shapes overlap, which one shows? **the circle**

18. Answer each question. PLAYER_SIZE is 40.

a) A ball has `x = 300`, `y = 375`, `vx = -2`, `vy = 6`, and `radius = 20`. Then `ball.update()` is called twice. Write x, y, vx, and vy after each call.

after	x	y	vx	vy
1 call	**298**	**381**	**-2**	**-6**
2 calls	**296**	**375**	**-2**	**-6**

b) A player has `rect.x = 10` and `rect.y = 350`. Then `player.move(-25, 15)` runs. What are `rect.x` and `rect.y` now? `rect.x` **0** `rect.y` **360**

c) A rock has `x = 100`, `y = 395`, `radius = 20`, and `speed = 5`. Then `rock.update()` runs. Does `update` call `reset`? Write Yes or No. **No**

d) What does `rock.get_rect()` return for that rock now? **Rect(80, 380, 40, 40)**

e) A player's `rect` is `Rect(110, 360, 40, 40)`. For that rock, is `player.rect.colliderect(rock.get_rect())` True or False? **True**