EECS 12 Review

Generic

1. print function

```
print(a, b, c, sep="", end="")
```

2. General input

```
num = eval(input("Enter Number:"))
```

3. Definite loop run for n times

```
for i in range(n):
```

4. Operators

- + - * \/
- ** (power)
- // (integer division)
- % (modulo)
- abs (absolute value)

5. Square root

```
import math
math.sqrt(4)
```

6. If statement

```
if <condition>:
    <body>
```

7. Condition Operator

```
<   <=   ==   >=   >   !=
```

8. Data type

- int()
- string()
- float()
- boolean()

9. Round function

```
round(3.9)
>>>4
round(3.9154, 2)
>>>3.92
```

10. Simultaneous Assignment

```
<var1?, <var2> = <expr1>, <expr2>
```

11. General definite loop

```
for i in [a, b, c, d]:
    <expr>
```

12. Range function

- return a list
- range(start, n, step)
    1. Find the value of 'start'
    2. Find the value of 'step'
    3. if step>0 op='<', else op='>'
    4. Check if "start op n" is true, output start, else loop
    5. Go to step 3

6. Stop a loop

```
break(go to the outer layer)
```

14. List interfaces

- List.append(x)

  append the element x to the list
- *

  zeros = [0] * 50
  repeat 50 times
- len(list)

  return the length of the list
- slice the list

  lst = [1, 2, 3, 4]
  lst[1:3] = [2, 3]

- in

```
x in list
```

return a boolean value that whether x is in list

- delete

```
del myList[1:3]
```

15. String interfaces

- +

  connect two string
- *

  repeat the string
- string[index]

  indexing
- string[begin:end]

  slicing
- len()

  length
- split

```
>>>'32,24,25,57'.split(',')
['32','24','25','57']
```

- \n

  Enter: next line
- string.find(s)

  return the index when s first occurs
- string.count(s)

return the number of the occurence of s

- string.replace(old, new)

  return a string that the old string was replaced by the new string

  Attention: you have to use "s = s.replace(old, new)" in order to change s

16. Two dimensional list

```
>>>a = [[1, 2], [3, 4]]
>>>a[1][1]
4
```

17. File processing

```
file = open('filename')

# return the string of that file
file.read()

# return the next line of the file
file.readline()

# return the list of all lines of the file
file.readlines()

# close the file
file.close()
```

18 ASCII table

```
# @param: the index of the char
# @return: return the char of the index of num
chr(num)

# @param: a char
# @return: return the index of that char
```

19. type function

```
# @return: return the type of x
type(x)
```

20. String formatting
    (see slide for details...)
21. Functions

```
def functioname(param1, param2....):
    functions
    return ret1, ret2...
    # when return the function exits
```

22. The function does not have access to the variable that holds the actual parameter. Python pass the parameters by value.
23. If the value of the variable is a mutable object(like a list or graphic), then changes to the state of the object will be visible to the calling program.

Graphics

1. create window

```
>>> from graphics import *
>>> win = GraphWin()
>>> win.close()
# The default GraphWin is 200 pixels tall by 200 pixels wide and origin(0, 0) is in the upper
left corner, and the lower right corner is (199, 199)
```

2. sample

```
def main():
    # create a window named My Circle with width 100 and height 100.
    win = GraphWin("My Circle", 100, 100)

    # create a circle object and draw it on the window
    c = Circle(Point(50, 50), 10)
    c.draw(win)

    # when the mouse is clicked, close the window
    win.getMouse()
    win.close()

main()
```

3. types of graphic operations

- creating graphics objects by their class names
- displaying objects
- changing objects: position, color
- getting user clicks
- getting user typing entry
- defining custon coordinates

4. Graphical Objects

- GraphWin
- Point
- Circle
- Oval

- Line
- Text
- Rectangle

5. creating graphical objects

```
varObj = Class(initial values)
```

6. using graphical objects

```
# <object>.<method-name>(<param1>, <param2>, ...)
>>> p = Point(50, 60)
>>> p.getX()
50
```

7. set color

```
# set the fill color of a object
<obj>.setFill('<color>')

# set the outline color of a object
<obj>.setOutline('<color>')

# set to the custom color
<obj>.setFill(color_rgb(255, 0, 0))
```

8. Set coordinate

```
# @params: four parameters that composed two coordinates of the lower-left and upper-right
corners.
```

9. Move

```
# move the object dx unit in the x direction and dy in the y direction
<obj>.move(dx, dy)
```

10. Get mouse clicks

```
from graphics import *
win = GraphWin('Click me!')
p = win.getMouse()

# p is the Point that you clicked
print("You clicked at:", p.getX(), p.getY())
```

11. delay

```
from time import *

# sleep for 0.1 second
sleep(0.1)
```

12 Moving object sample code

```
win = GraphWin(“Circle Window”, 400, 400)
x = win.getMouse()
x.draw(win)
c = Circle(x, 100)
c.draw(win)
for in in range(10):
    c.move(100/10,100/10)
    sleep(0.1)
p = win.getMouse()
win.close()
```

12. Handling Textual input

```
input = Entry(Point(2, 3), 5)
input.setText("0.0")
input.draw(win)
str = input.getText()
```

13. Showing pictures
    Note: only gif and ppm could be displayed

```
win = GraphWin("Bugs", 800, 800)
initPoint = Point(50, 50)
bug = Image(initPoint, "bug.gif")
bug.draw(win)
```

Class & Objects

1. Example of Constructor

```
circ = Circle(Point(0,0), 20)
# circ is a instance
```

2. Example of instance's method

```
circ.draw(win)
```

3. Class Definition

```
class <class-name>:
        <method-definitions>
```

4. Class Definition example

```
class Student:
        # __init__ is the constructor
        def __init__(self, name. hours, qpoints):
                self.name = name
                self.hours = float(hours)
                self.qpoints = float(qpoints)

        # The self parameter is required for a method calling self.<var>
        def getName(self):
                return self.name

        def getHours(self):
                return self.hours

        def getQPoints(self):
                return self.qpoints

        def gpa(self):
                return self.qpoints/self.hours

        # a static function which not calling self.<var> does not require self as a parameter of
the method
        def makeStudent(infoStr):
                name, hours, qpoints = infoStr.split()
                return Student(name, hours, qpoints)
```

5. Examples using class

```
>>> stu = Student("abc", 1, 1)
>>> print(stu.getName())
abc
>>> print(stu.gpa())
1
>>> stu2 = makeStudent("bcd 1 1")
>>> print(stu2.getHours())
1
```

Searching Algorithms

Linear Search

1. Better performance for small data
2. Python use Linear search as its built-in searching method

Binary Search

1. REQUIRE THE DATASET TO BE SORTED
2. Better performance for large data
3. If the data is unsorted, it must be sorted first