Intro to Programming (No Prior Experience)

Class 12 – Module 6 Review

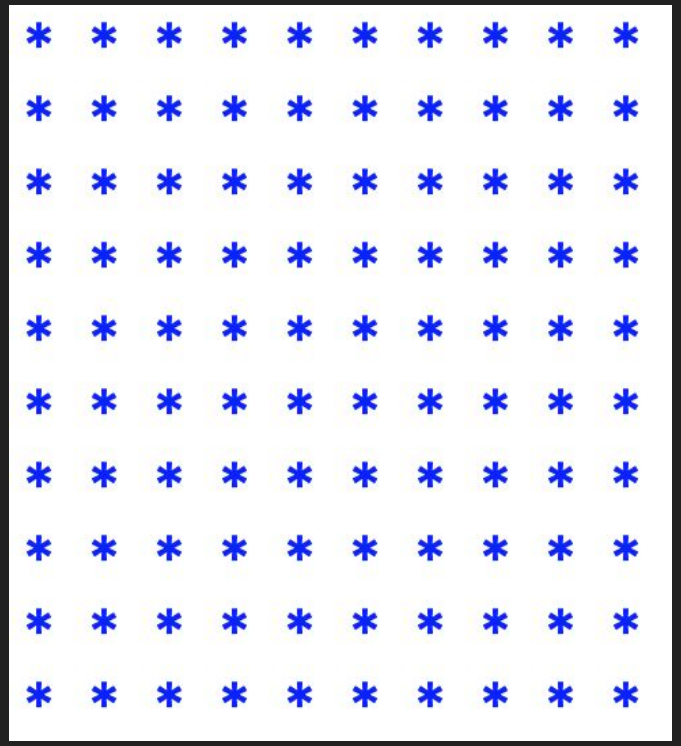
Emily Zhao
T/R 4:55PM–6:10PM

Agenda

- Review For Loops
- Review Module 6 + Quiz 6
- Practice Problems

“For” loop review

For Loops Review



1. Generate a 10x10 grid of asterisks using **1** for loop
2. Change your code so that it can generate an any number by any number grid
3. Generate the grid again, this time using **nested** for loops

```
rows = 10
cols = 10

'''
# Single for loop
for r in range(0, rows)
    print("* " * cols)

# Nested for loop
for c in range(0, cols):
    for r in range(0, rows):
        print("*", end=" ")
    print()
'''

# Checkerboard
rows = 10
cols = 10

for c in range(0, cols):
    for r in range(0, rows):
        # if the sum of the row # and column #
        # is even, then draw one symbol
        if (c + r) % 2 == 0:
            print("@", end=" ")
        else:
            print("#", end=" ")
    print()
```

for / else

- **for** loops also have an **else** clause
- The else clause executes after the loop completes normally.
- This means that the loop did not encounter a break statement.

```
for x in range(1, 4):
    print(x)
else:
    print("Out of the loop")
```

```
1
2
3
Out of the loop
```

for / else

```
for x in range(1, 4):
    print(x)
else:
    print("Out of the loop")
```

```
1
2
3
Out of the loop
```

```
for x in range(1, 4):
    print(x)
    if x == 2:
        break
else:
    print("Out of the loop")
```

```
1
2
```

for / else

```
user_input = "kiwi"

for fruit in ["apple", "banana", "peach"]:
    if fruit == user_input:
        print("Your fruit is in the list!")
        break
else:
    print("We could not find your fruit.")
```

```
We could not find your fruit
```

Can you rewrite your prime number finder using for/else?

```
# Program to check if a number is prime or not

num = 407

# To take input from the user
#num = int(input("Enter a number: "))

# prime numbers are greater than 1
if num > 1:
   # check for factors
   for i in range(2,num):
       if (num % i) == 0:
           print(num,"is not a prime number")
           print(i,"times",num//i,"is",num)
           break
   else:
       print(num,"is a prime number")
```

Functions

Functions

- A function is a group of statements that exist within a program for the purpose of performing a specific task
- Since the beginning of the semester we have been using a number of Python's built-in functions, including:
  - print()
  - range()
  - len()
  - random.randint()
  - ... etc

3 reasons to use functions

1. Organize your code
2. Reuse your code
3. Collaborate with others

Defining Functions

- Functions, like variables must be named and created before you can use them
- The same naming rules apply for both variables and functions
  - You can’t use any of Python’s keywords
  - No spaces
  - The first character must be A-Z or a-z or the “_” character
  - After the first character you can use A-Z, a-z, “_” or 0-9
  - Uppercase and lowercase characters are distinct

Defining functions

```
def myfunction():
        print ("Printed from inside a function")

# call the function
myfunction()
```

Some notes on functions

- When you run a function you say that you “call” it
- Once a function has completed, Python will return back to the line directly after the initial function call
- Functions must be defined before they can be used. In Python we generally place all of our functions at the beginning of our programs.

Flow of Execution with Functions

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
Hi there!
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
Hi there!
I’m a function!
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
Hi there!
I’m a function!
And now we’re done
```

Multiple Functions

Multiple functions

Code

```
def hello():
    print(“Hello there!”)

def goodbye():
    print(“See ya!”)

hello()
goodbye()
```

Output

Multiple functions

Code

```
def hello():
    print(“Hello there!”)

def goodbye():
    print(“See ya!”)

hello()
goodbye()
```

Output

```
Hello there!
See ya!
```

Multiple functions

Code

```
def main():
    print(“I have a message for you”)
    message()
     print(“Goodbye!”)

def message():
    print(“The password is ‘foo’”)

main()
```

Output

Multiple functions

Code

```
def main():
    print(“I have a message for you”)
    message()
     print(“Goodbye!”)

def message():
    print(“The password is ‘foo’”)

main()
```

Output

```
I have a message for you
The password is ‘foo’
Goodbye!
```

Local Variables

Local Variables

- Functions are like “mini programs”
- You can create variables inside functions just as you would in your main program

```
def bugs():
    numbugs = int(input('How many bugs? '))
    print(numbugs)

bugs()
```

Local Variables

- However, variables that are defined inside of a function are considered “local” to that function.

- This means that they only exist within that function. Objects outside the “scope” of the function will not be able to access that variable

```
1 def bugs():
2     numbugs = int(input('How many bugs? '))
3     print(numbugs)
4
5 bugs()
6
7 print(numbugs)
```

```
How many bugs? 3
3
Traceback (most recent call last):
  File "/Users/emilyzhao/Documents/test.py", line 7,
module>
    print(numbugs)
NameError: name 'numbugs' is not defined
```

Local Variables

- Different functions can have their own local variables that use the same variable name

- These local variables will not overwrite one another since they exist in different “scopes”

Code

```
def newjersey():
  numbugs = 1000
  print (“NJ has”, numbugs, “bugs”)

def newyork():
  numbugs = 2000
  print (“NY has”, numbugs, “bugs”)

newjersey()
newyork()
```

Output

```
NJ has 1000 numbugs
NY has 2000 numbugs
```

Passing Arguments to a Function

Passing Arguments to a Function

- Sometimes it’s useful to not only call a function but also send it one or more pieces of data as an argument

- This process is identical to what you’ve been doing with the built-in functions we have studied so far
  - `x = random.randint(1,5)    # send 2 integers`
  - `y = len(‘Craig’)        # send 1 string`

Passing Arguments to a Function

```
def square(num):
    print (num**2)  # num assumes the value of the
                    # argument that is passed to
                    # the function (5)

square(5)
```

Passing Multiple Arguments to a Function

```
def average(num1, num2, num3):
    sum = num1+num2+num3
    avg = sum / 3
    print (avg)

average(100, 90, 92)
```

Argument Mechanics

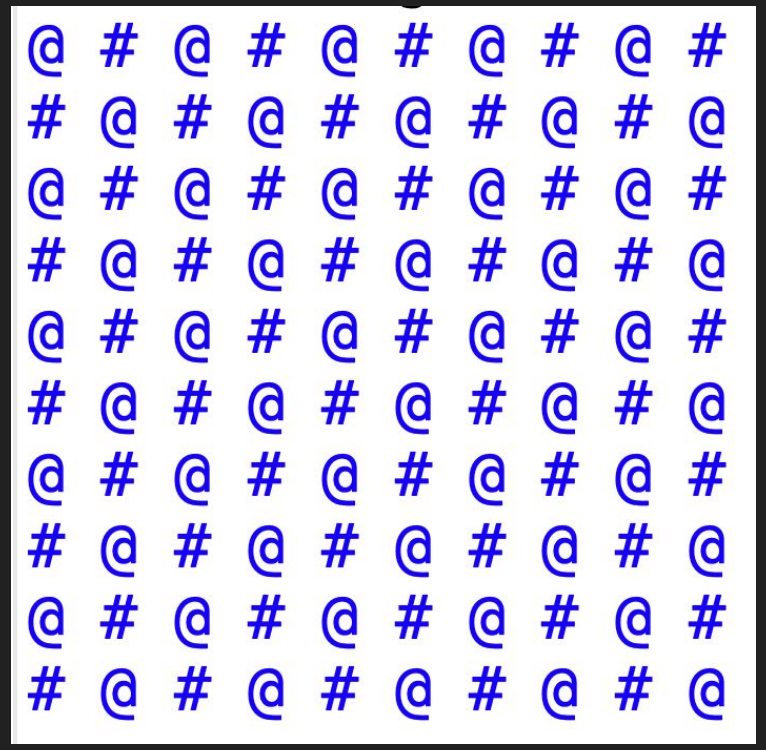
- When we pass an argument to a function in Python we are actually passing it's "value" into the function, and not an actual variable

```
def change_me(v):
    print ("function got:", v)
    v = 10
    print ("argument is now:", v)

myvar = 5
print ("starting with:", myvar)
change_me(myvar)
print ("ending with:", myvar)
```

```
starting with: 5
function got: 5
argument is now: 10
ending with: 5
```

Programming Challenge



Convert our earlier checkerboard code into a function that accepts three parameters – grid size, first character, second character

Global Variables

Global Variables

- When you create a variable inside a function we say that the variable is “local” to that function
- This means that it can only be accessed by statements inside the function that created it
- When a variable is created outside all of your functions it is considered a “global variable”
- Global variables can be accessed by any statement in your program file, including by statements in any function
- All of the variables we have been creating so far in class have been global variables

```
name = 'Emily'

def showname():
    print ("Function:", name)

print ("Main program:", name)
showname()
```

```
Main program:
Function:
```

```
name = 'Emily'

def showname():
    name = 'Charlie'
    print ("Function:", name)

print ("Main program:", name)
showname()
```

```
Main program: Emily
Function: Charlie
```

Global Variables

- If you want to be able to change a global variable inside of a function you must first tell Python that you wish to do this using the “global” keyword inside your function

```
name = 'Emily'

def showname():
    global name
    print("Function 1:", name)
    name = 'Charlie'
    print ("Function 2:", name)

print ("Main program:", name)
showname()
print ("Main program 2:", name)
```

```
Main program: Emily
Function 1: Emily
Function 2 Charlie
Main program 2: Charlie
```

Global Variables

- Global variables can make debugging difficult
- Functions that use global variables are generally dependent on those variables, making your code less portable
- With that said, there are many situations where using global variables makes a lot of sense.

Value Returning Functions

Value Returning Functions

- Value returning functions are functions that return a value to the part of the program that initiated the function call
- They are almost identical to the type of functions we have been writing so far, but they have the added ability to send back information at the end of the function call
- We have secretly been using these all semester!
    - `somestring = input(“Tell me your name”)`
    - `somenumber = random.randint(1,5)`

Writing your own value returning functions

- You use almost the same syntax for writing a value returning function as you would for writing a normal function
- The only difference is that you need to include a “return” statement in your function to tell Python that you intend to return a value to the calling program
- The return statement causes a function to end immediately.
  It’s like the break statement for a loop.
- A function will not proceed past its return statement once encountered.
  Control of the program is returned back to the caller.

```
def myfunction(arg1, arg2):
    statement
    statement
    …
    statement
return expression

# call the function
returnvalue = myfunction(10, 50)
```

```
def testfunction():
    x = 5
    y = 10
    return x, y

p, q = testfunction()
```

IPO Notation

- As you start writing more advanced functions you should think about documenting them based on their Input, Processing and Output (IPO)

- Example:

```
# function: add_ages
# input: age1 (integer), age2 (integer)
# processing:  combines the two integers
# output: returns the combined value

def add_ages(age1, age2):
      sum = age1+age2
      return sum
```

Homework

— Assignment #5 (due Tuesday)