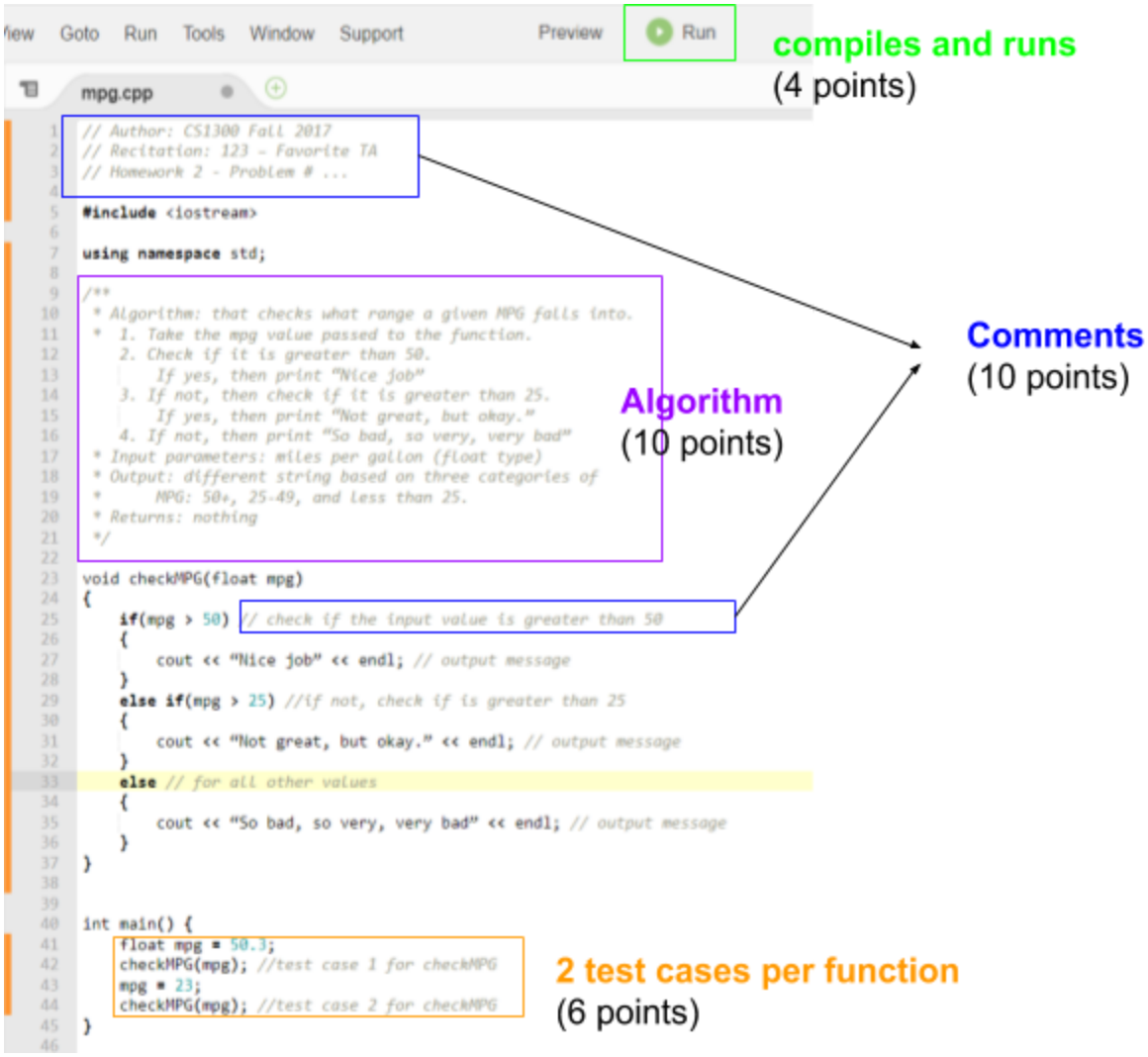
CSCI 1300 - Spring 2018
Instructor: Fleming/Gupta
Recitation 13
Due Saturday, April 21st, by 6:00pm

- All components (Cloud9 workspace and zip file) must be completed and submitted by **Saturday, April 21st 6:00 pm** for your solution to receive points.
- Recitation attendance is required to receive credit.
- **Develop in Cloud9:** For this recitation assignment, write and test your solution using Cloud9.
- **Submission:** Both steps must be fully completed by the submission deadline for your homework to receive points. Partial submissions will not be graded.
    1. Make sure your Cloud9 workspace is shared with your TA: Your recitation TA will review your code by going to your Cloud9 workspace. TAs will check the last version that was saved before the submission deadline.
        - Create a directory called *Rec13* and place all your file(s) for this assignment in this directory.
    2. **Submit a zip file to Moodle:** After you have completed all the questions and checked them on Moodle, you must submit a zip file with all the .py file(s) you wrote in Cloud9. Submit this file going to *Rec 13 (File Submission)* on moodle.

- 4 points for code compiling and running
- 6 points for two test cases for each function
- 10 points for an algorithm description for each function
- 10 points for comments
    - required comments at top of file

```
# CS1300 Spring 2018
# Author:
# Recitation: 123 – Favorite TA
# Cloud9 Workspace Editor Link: https://ide.c9.io/…
# Recitation 13 - Problem # ...
```

    - comments for functions / test cases


iew Goto Run Tools Window Support
Preview
Run
mpg.cpp
// Author: CS1300 Fall 2017
// Recitation: 123 - Favorite TA
// Homework 2 - Problem # ...
#include <iostream>
using namespace std;
/**
* Algorithm: that checks what range a given MPG falls into.
* 1. Take the mpg value passed to the function.
2. Check if it is greater than 50.
If yes, then print "Nice job"
3. If not, then check if it is greater than 25.
If yes, then print "Not great, but okay."
4. If not, then print "So bad, so very, very bad"
* Input parameters: miles per gallon (float type)
* Output: different string based on three categories of
* MPG: 50+, 25-49, and less than 25.
* Returns: nothing
*/
void checkMPG(float mpg)
{
if(mpg > 50) // check if the input value is greater than 50
{
cout << "Nice job" << endl; // output message
}
else if(mpg > 25) //if not, check if is greater than 25
{
cout << "Not great, but okay." << endl; // output message
}
else // for all other values
{
cout << "So bad, so very, very bad" << endl; // output message
}
}
int main() {
float mpg = 50.3;
checkMPG(mpg); //test case 1 for checkMPG
mpg = 23;
checkMPG(mpg); //test case 2 for checkMPG
}
compiles and runs
(4 points)
Comments
(10 points)
Algorithm
(10 points)
2 test cases per function
(6 points)

Python

From this point on in the semester we will be working with Python (specifically Python 3).

A great resource for learning Python is, *Learning Python the Hard Way*: https://learnpythonthehardway.org/book

You will have to save your files as **.py** (for **py**thon) instead of .cpp.

You can click on the "run" button to run Python programs in Cloud9. If you wish to execute a python program on the command line, you can do:

```
python fileName.py
```

Indentation

In Python the delimiter is a indentation of the code itself. No curly braces will be used to mark where the code starts and stops and for any loops, if-else statements, or functions there is not an explicit begin or end without the indentation. There are no explicit braces, brackets, or keywords. This means that whitespace is significant, and must be consistent.

“Code blocks” or sections of code are defined by their indentation. Code blocks means sections of code for functions, if statements, for loops, while loops, and so forth. Indenting starts a block and is ended when the indentation moves back a tab.

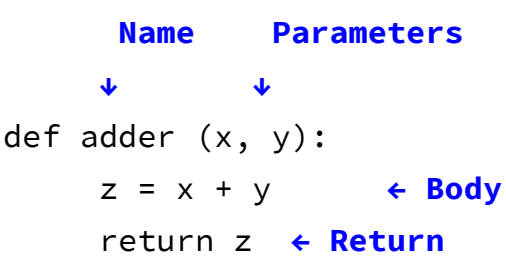
Pound/Hash (#) is used for single line comments in Python. Use """ or ''' (three double or single quotes) for multi line comments.

For example:

```
Block 1         #statements in Block 1 are Executed
      Block 2       #statements in Block 2 are Executed
            Block 3 #statements in Block 3 are Executed
      Block 2       #Remaining statements in Block 2 are
Executed
Block 1         #Remaining statements in Block 1 are Executed
```

Functions in Python

A function has a few key components: the name, the parameters, the body and the return value. Let's take a look at a simple example of a Python function.



Function Example

This example defines a function called adderTwo that takes two parameters and returns their sum. The adderTwo function is called from another function called main.

```
def adderTwo(numOne, numTwo):
	print(numOne, numTwo)
	numOne = numOne + numTwo
	print(numOne)
	return numOne

def main():
	X = 3
	Y = 4
	print(X,Y)
	Z = adderTwo(X,Y)
	print(X, Y, Z)

# this must be included for the main function to be called when the program is run
if __name__ == "__main__":
	main()
```

Loops in Python: For and While

Loops allow us to run a chunk of code multiple times. If we know how many times we'll

need to run the code, we typically use a for loop. When we don't know how many times we need to run the code, we usually use a while loop.

For Loops

The basic structure of a for loop in Python looks like this:

```
for i in range(startNum, endNum):
      # do something using i
```

Here, i is the dummy variable and range(startNum, endNum) is the iterable. The dummy variable takes on the values of the iterable, beginning with startNum and up to, **but not including**, endNum. Each loop, i will increment by 1. For example, if we enter the following code:

```
for i in range(0, 5):
      print(i)
```

We should see the following output:

```
0
1
2
3
4
```

Alternatively, we can define range with a single parameter: range(endNum) By default, the dummy variable will begin at 0 and iterate up to, but not including, endNum.

If we want to have i increase by some number other than 1, we can specify this number as the 3rd parameter in range:

```
for i in range(0, 5, 2):
      print(i)
```

The output:

```
0
2
4
```

Of course we can do things far more complex than just printing a single variable. Try writing a loop that outputs the even numbers between 0 and 10.

While loops and user input

A while loop in python uses the following syntax:

```
while(condition):
      # do something
```

As long as condition is true, the code within the while loop will run. Consider the following code:

```
userChoice = 1
while(userChoice != 0):
      userChoice = int(input("Do you want to see this question again? Press 0 for
no, any other number for yes"))
```

Entering '0' will terminate the loop, but any other number will cause the loop to run again.

Note how we have to initialize the condition before the loop starts. Setting `userChoice` equal to 1 ensures that the while loop will run at least once.

Also note that we must "cast" the input to type `int`. By default, `input()` returns a string value. Consequently, we must turn it into an integer to have `userchoice != 0` evaluated within the while condition.

Converting For into While

It's possible to use a while loop instead of a for loop. Let's rewrite the second for loop example from above as a while loop:

```
i = 0
while i < 5:
      print(i)
      i = i + 2
```

This accomplishes the same as output as above, but with more lines of code. Notice how you must manually initialize `i` to equal 0 and then manually increment `i` by 2.

Recitation Activity

Question 1

Write a function **print_param** that prints out whatever is passed in as argument. The function should take in one parameter and should not return anything.

Question 2

Write a function **odd_or_even** that takes in one integer and prints out whether the integer is even or odd. (e.g. if the input is even, print the string 'even', and if the number is odd, print the string 'odd') The function should take in one parameter and should not return anything.

Question 3

Write a function **fizz_buzz** that loops from 0 up to a number *n*, printing the number or a word at each iteration.

- If the current number is divisible by 3, print the word “Fizz”
- If it is divisible by 5, print “Buzz”
- If it is divisible by 3 and 5 print "FizzBuzz"
- Otherwise, print the number itself. (If the number is 0, also print the number itself.)

The function should take one parameter *n* and should not return anything.

Question 4

Write a function **oneSum** which takes one input, n, and finds the sum:

1 + 11 + 111 + 1111 + ... up to *n* digits of 1.

For example:

```
>>>print (oneSum( 1 ))

1
```

Which is 1 = 1.

```
>>> print (oneSum( 2 ))
12
```

Which is 1 + 11 = 12.

```
>>> print (oneSum( 3 ))
123
```

Which is 1 + 11 + 111 = 123.

Hint: think about each element of the sum (1, 11, 111, for example) represented as a sum of powers of 10:

111 = 10^0 + 10^1 + 10^2 = 1 + 10 + 100