Pure Problems

University of Wyoming COSC 1010

Adapted from: *Think Like a Programmer* By V. Anton Spraul

Pure Puzzles

- We wills tart looking at using code to solve different problems
- The goal here is to solve the problems, not to worry about the code to do so
- Once you figure out what you want to do, you can translate your thoughts into code

Output Patterns

- For this lecture we will work through three main problems
- But, we will utilize the problem division and reduction techniques to turn these into sub problems
- We will look at a series of problems to output specific patterns

Half a Square

- The goal of this is to write a program to produce the following pattern

```
#####
####
###
##
#
```

Half a Square

- But, problems have constraints!
- In this case you can only use the following print statements:
  - `print()`, printing a newline character
  - `print('#', end="")`, printing a single hash without a new line
  - Each of these can only occur once in your program

- If we ignored constraints this problem would be easy to solve
- But as we can only use the two print statements to output to the terminal, it becomes more difficult

Half a Square

- As we have the constraints, and know we need to have multiple outputs we logically turn to loops
- You may already have a solution in your head, but what if you don't?
- How can rwe reduce the problem so it is easy to solve
- What if we started with a whole square?

A Square

- Can we write a program that outputs a full square, rather than the half we want?
- This is a far more trivial of a problem
- But, it gives us a starting place to solve our half a square problem

In [ ]:
```
for i in range(0,5):
    print("#",end="")
print()
```

```
#####
```

A Square

- That is a basic reduction
- It shows us how to print `######` out easily using a single loop
- We can then add to out code to print the full square shape
- We just need to repeat our current pattern five times
- Seems like its time for another loop!

In [ ]:
```
for j in range(0,5):
    for i in range(0,5):
        print("#",end="")
    print()
```

```
#####
#####
#####
#####
#####
```

A Square

- We placed our first code in a second loop
- Now our original code repeats 5 times
- So, we are getting closer to the proper solution for our ultimate problem
- But, how do we modify the code we have here to produce the half square?

A Square

- We know we want five rows, so our outer loop is correct
- The issue is our inner loop then, which prints to columns of our output
- What we need then is a way to adjust the number of of symbols produced by the inner loop
- We need 5 in the first row, 4 in the second ...

A Square

- We'll start by making another reduced program to experiment
- It is often easiest to work on troublesome portions of a problem in isolation
- We will forget hash symbols for a moment and just focus on numbers
- Write a program that displays the numbers 5-1, in that order
- We want each number on a separate line

In [ ]:
```
i = 5
while i > 0:
    print(i)
    i -= 1
```

```
5
4
3
2
1
```

In [ ]:
```
rng = list(range(1,6))
rng.reverse()
print(rng)
for i in rng:
    print(i)
```

```
[5, 4, 3, 2, 1]
5
4
3
2
1
```

A Square

- We need an *expression* of some kind that is 5 when row 1
  - 4 when row is 2 ....
- Essentially we need an expression that decreases as row increases
- Perhaps we could utilize a minus sign that corresponds with row?

In [ ]:
```
for j in range(0,5):
    for i in range(0,5):
        print("#",end="")
    print()
```

```
#####
#####
#####
#####
#####
```

A Square

row	j	desired num of #	sum of j+dn#
1	0	5	5
2	1	4	5
3	2	3	5
4	3	2	5
5	4	1	5

A Square

- Once you list out everything hopefully you can see something helpful
  - The number of `#` we want on each line, dn# can be determined by `dn# = 5 -j`
- This is excellent, as `j` is part of our upper loop, which we already decided was what we needed
- So how can we use our newly discovered formula to produce the correct output?

In [ ]:
```
for j in range(0,5):
    for i in range(0,5-j):
        print("#",end="")
    print()
```

```
#####
####
###
##
#
```

Half a Square

- TADA!
- This works out nicely, and if it hadn't the changes likely would have been minor
- Using the reduction technique requires more steps to get from the problem to a solution
  - But, each step is easier to accomplish
- it is kind of like using pulleys, you may have to pull the rope farther but it will be easier to do so

A Sideways Triangle

- Now we are going to write a program using the same outputs to produce the following:

```
#
##
###
####
###
##
#
```

A Sideways Triangle

- We don't need to go through the same amount of steps as last time
- Largely as this problem is analogous to the "half a square" problem we just did
- So, what can we learn from our previous problem that we can apply to the current problem?
- Remember to always start with what you know

A Sideways Triangle

- So what do we know?
  - How to display a row of symbols of a particular length with a loop
  - How to display a series of rows using nested loops
  - How to create a varying number of symbols in each row
  - How to discover the correct expression through experimentation

A Sideways Triangle

- Most of the mental work has already done, while solving the previous problem
- We can start by experimenting once again
- In our previous problem subtracting the row from the larger number worked well
  - But that isn't directly applicable here as we increase and decrease the number of elements
- We can break things into a table again, and some starting code

In [ ]:
```
for i in range (0,7):
    for j in range (0,4):
        print("#",end="")
    print()
```

```
####
####
####
####
####
####
####
```

A Sideways Triangle

row	i	dn #	4 - i
1	0	1	4
2	1	2	3
3	2	3	2
4	3	4	1
5	4	3	0

row	i	dn #	4 - i
6	5	2	-1
7	6	1	-2

A Sideways Triangle

- That isn't quite it
- But what adjustments can we make to maybe make it a bit closer to what we want
- What if we start i at 1, rather than 0?
- Would that help us at all?

A Sideways Triangle

row	i	dn #	4 - i
1	1	1	3
2	2	2	2
3	3	3	1
4	4	4	0
5	5	3	-1
6	6	2	-2
7	7	1	-3

A Sideways Triangle

- Looking at that, how can we form our expression to get dn#
- For rows 1-4 our `dn# = 4 - (4-i)`
- But what about for rows 5-7?
  - Our dn# could be expressed as `dn# = 4 + (4-i)`
- Can we make our lives easier to express these in the same equation

A Sideways Triangle

- If we have the negative values of what we want to subtract, how can we get the positive?
- Well for rows 5-7 we could multiply `4-i` by `-1`
  - But then our equation wouldn't be generic enough to work for rows 1-4
- Really what we are looking for is the *absolute value* of `4-i`

A Sideways Triangle

---

row	i	dn #	4 - i	4 - abs(4-i)
1	1	1	3	1
2	2	2	2	2
3	3	3	1	3
4	4	4	0	4
5	5	3	-1	3
6	6	2	-2	2
7	7	1	-3	1

In [ ]:
```
for i in range (1,8):
    for j in range (0,4-abs(4-i)):
        print("#",end="")
    print()
```

```
#
##
###
####
###
##
#
```