Day 1

December 3, 2021

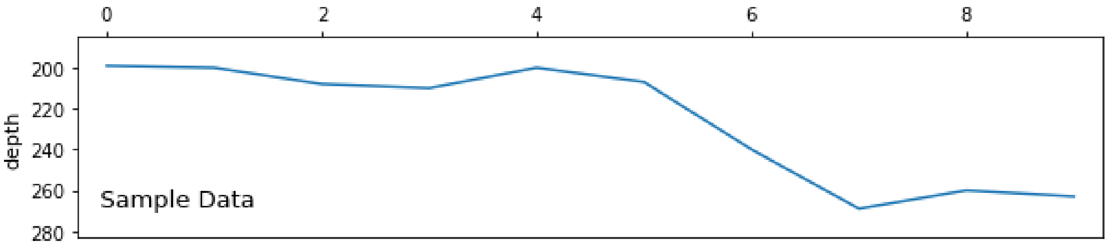
1 Advent of Code 2021 - Day 1

1.1 Setup

Parse and style.

[1]:
```
from libaoc.styles import *
plotStyle()

def plotSetup():
    plotInvertY(.1)
    plt.ylabel("depth")

_inputText = open('day1.input.txt').read()
_inputData = [int(x) for x in re.findall('\d+', _inputText)]

_sampleData = [
    199,
    200,
    208,
    210,
    200,
    207,
    240,
    269,
    260,
    263]

plotSetup(); plt.title('   Sample Data'); plt.plot(_sampleData)
```

[1]:
```
[<matplotlib.lines.Line2D at 0x2628762bee0>]
```

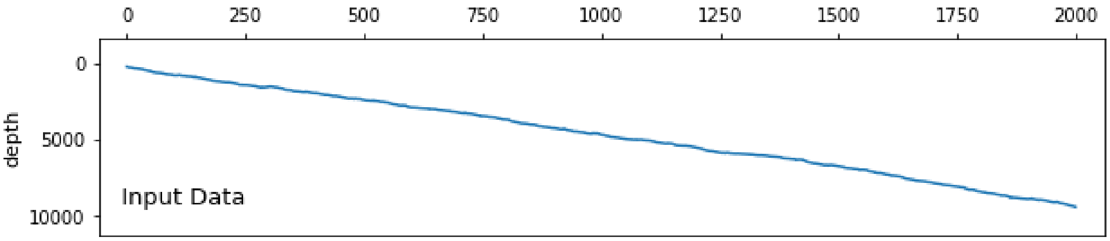


[2]:

```
plotSetup(); plt.title('    Input Data'); plt.plot(_inputData)
```

[2]:

```
[<matplotlib.lines.Line2D at 0x2628973f6a0>]
```



1.2 Solver

Part 1 and 2 can use the same solver:

- Walk list of ints
- Sum a moving window of given size
- Count instances where a sum is greater than the previous sum

[3]:

```
def solve(depths, window):
    last, count = 0, 0
    for depth in range(len(depths) - window):
        s = sum(depths[depth:depth+window])
        if s > last:
            count += 1
        last = s
    return count
```

1.3 Part 1

Window size = 1.

[4]:

```
def solve1(depths):
    return solve(depths, 1)
```

```
assert solve1(_sampleData) == 7

assert (s1 := solve1(_inputData)) == 1681
print(f"result = {s1}")
```

```
result = 1681
```

1.4 Part 2

Window size = 3.

[5]:

```
def solve2(depths):
    return solve(depths, 3)

assert solve2(_sampleData) == 5

assert (s2 := solve2(_inputData)) == 1704
print(f"result = {s2}")
```

```
result = 1704
```

Day 2

December 3, 2021

1 Advent of Code 2021 - Day 2

1.1 Setup

Parse and style.

[1]:

```
from libaoc.styles import *

plotStyle({'axes.prop_cycle': mpl.cycler('color', ['8dd3c7'])})

def plotSetup():
    plotInvertY(.1)
    plt.ylabel("depth")

_inputText = open('day2.input.txt').read()
_inputData = [(m, int(d)) for m, d in re.findall('(\w+) (\d+)', _inputText)]

_sampleData = [
    ('forward', 5),
    ('down', 5),
    ('forward', 8),
    ('up', 3),
    ('down', 8),
    ('forward', 2)]
```

1.2 Part 1

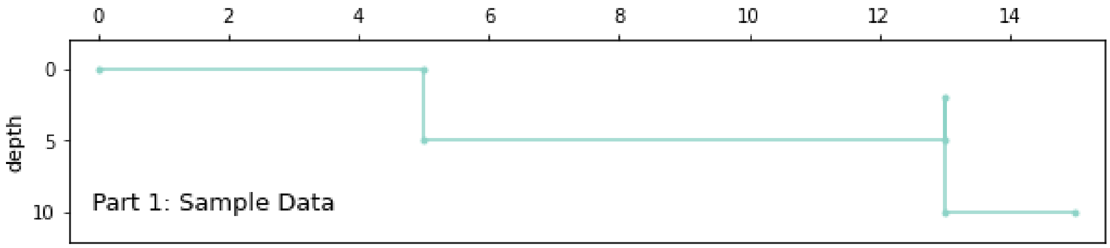
Algorithm:

- Adjust `x` based on forward movement
- Adjust `y` based on up/down movement
- Return `x*y`

[2]:

```
def solve1(moves):
    x, y = [0], [0]
    plotSetup()

    for move, dist in moves:
        dx, dy = 0, 0
        match move[0]:
            case 'f': dx = dist
            case 'd': dy = dist
            case 'u': dy = -dist
        x.append(x[-1] + dx)
        y.append(y[-1] + dy)

    plt.plot(x, y, marker='.')
    return x[-1] * y[-1]

assert solve1(_sampleData) == 150
plt.title('   Part 1: Sample Data')
```

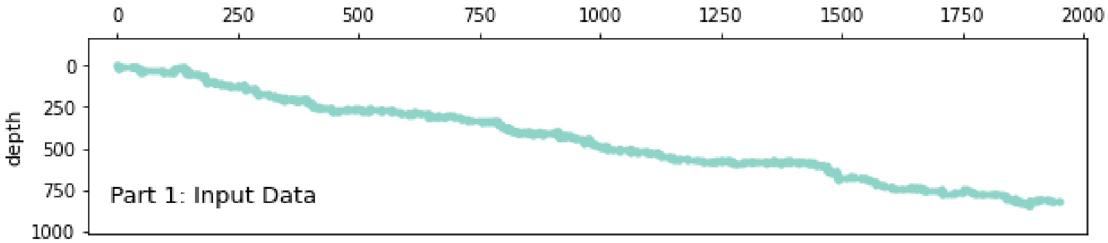
```
[2]: Text(0.0, 0.1, '   Part 1: Sample Data')
```



```
[3]: assert (s1 := solve1(_inputData)) == 1604850
     print(f"result = {s1}")
     plt.title('   Part 1: Input Data')
```

```
result = 1604850
```

```
[3]: Text(0.0, 0.1, '   Part 1: Input Data')
```



1.3 Part 2

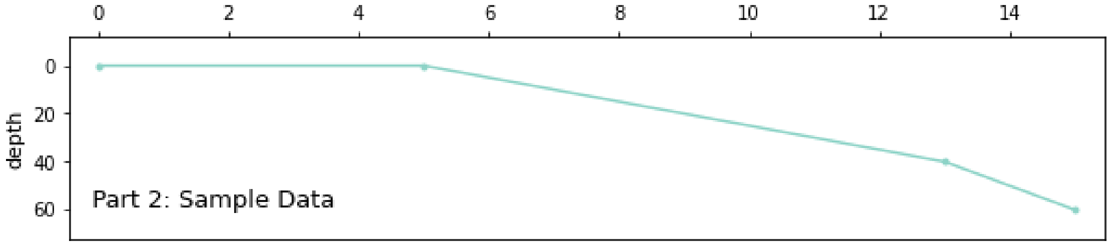
Make `aim` responsible for depth changes.

- Adjust `x` based on forward movement

- Adjust y based on aim during forward movement.
- Adjust aim based on up/down movement.
- Return x*y

[4]:
```
def solve2(moves):
    x, y, a = [0], [0], 0
    plotSetup()

    for move, dist in moves:
        match move[0]:
            case 'f':
                x.append(x[-1] + dist)
                y.append(y[-1] + a * dist)
            case 'd': a += dist
            case 'u': a -= dist

    plt.plot(x, y, marker='.')
    return x[-1] * y[-1]

assert solve2(_sampleData) == 900
plt.title('   Part 2: Sample Data')
```

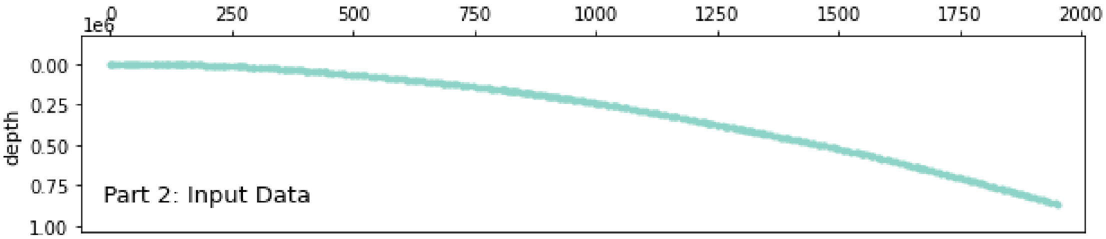
[4]:
```
Text(0.0, 0.1, '   Part 2: Sample Data')
```



[5]:
```
assert (s2 := solve2(_inputData)) == 1685186100
print(f"result = {s2}")
plt.title('   Part 2: Input Data')
```

```
result = 1685186100
```

[5]:
```
Text(0.0, 0.1, '   Part 2: Input Data')
```


0
250
500
750
1000
1250
1500
1750
2000
1e6
0.00
0.25
0.50
0.75
1.00
depth
Part 2: Input Data