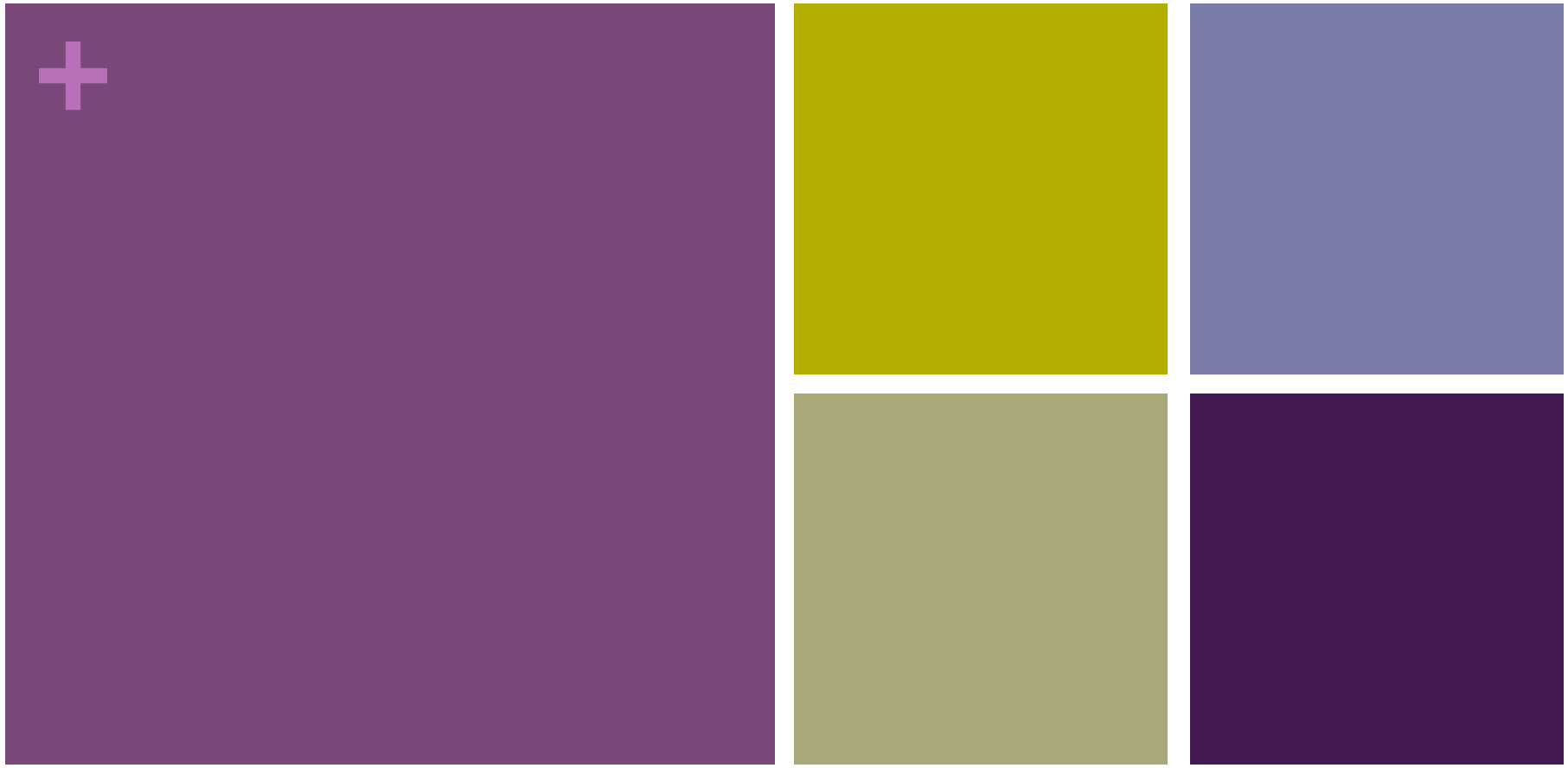


The List Data Structure

Variables vs. Lists

- So far we have been working with variables, which can be thought of as "buckets" that hold a particular piece of data
- Variables can only hold one piece of data at a time. Example
  - `x = 5`
  - `y = 5.0`
  - `z = 'hello'`
  - `q = True`
- However, there are times when we need to keep track of multiple pieces of data at the same time, and a single variable is limited to holding just one thing at a time

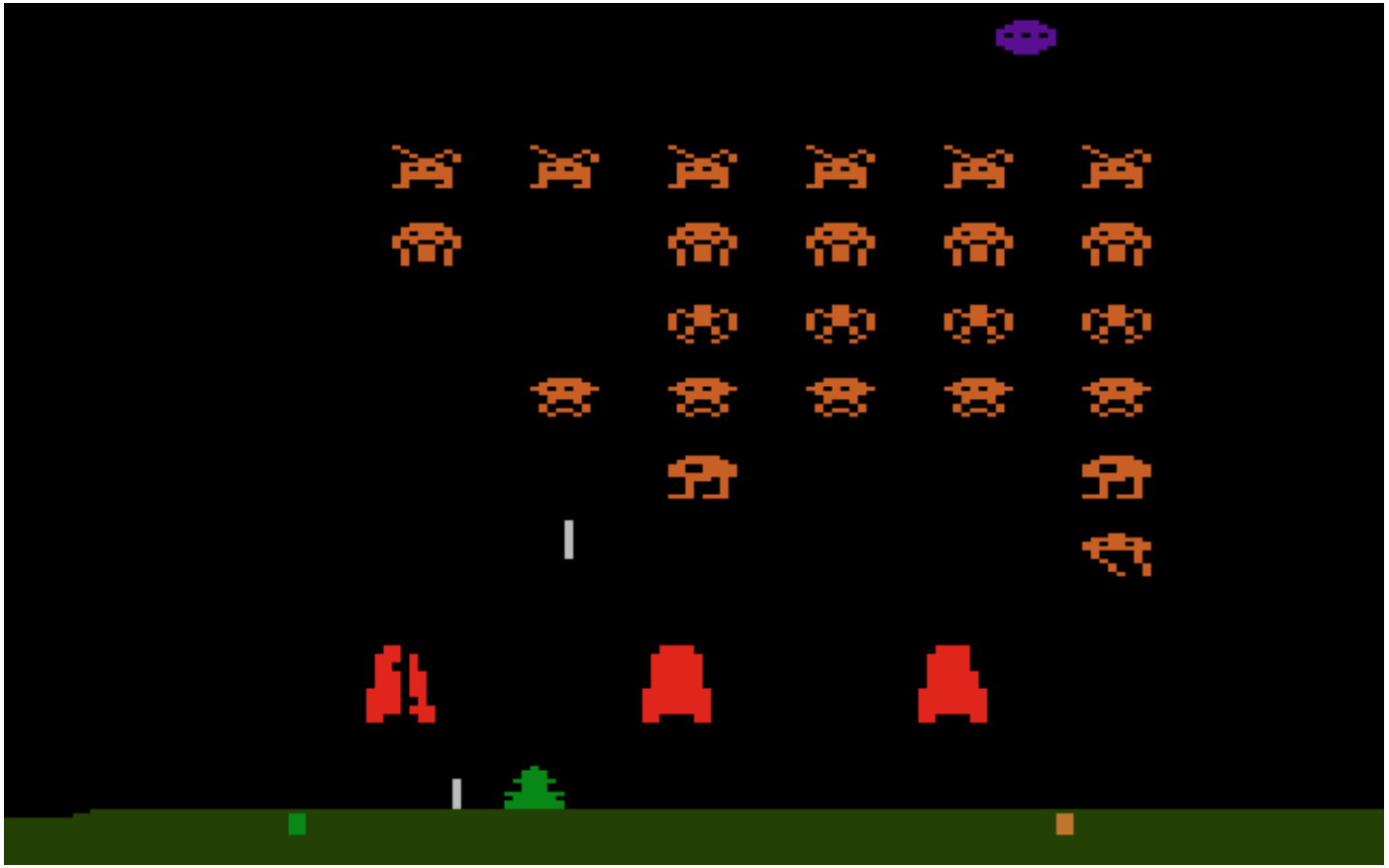
Lists

- Lists are considered a "sequence" object. Sequence objects have the ability to hold multiple pieces of data at the same time.
- We can use a single sequence variable to hold any number of values.
- In most programming languages we call these "arrays." In Python we call these "lists."

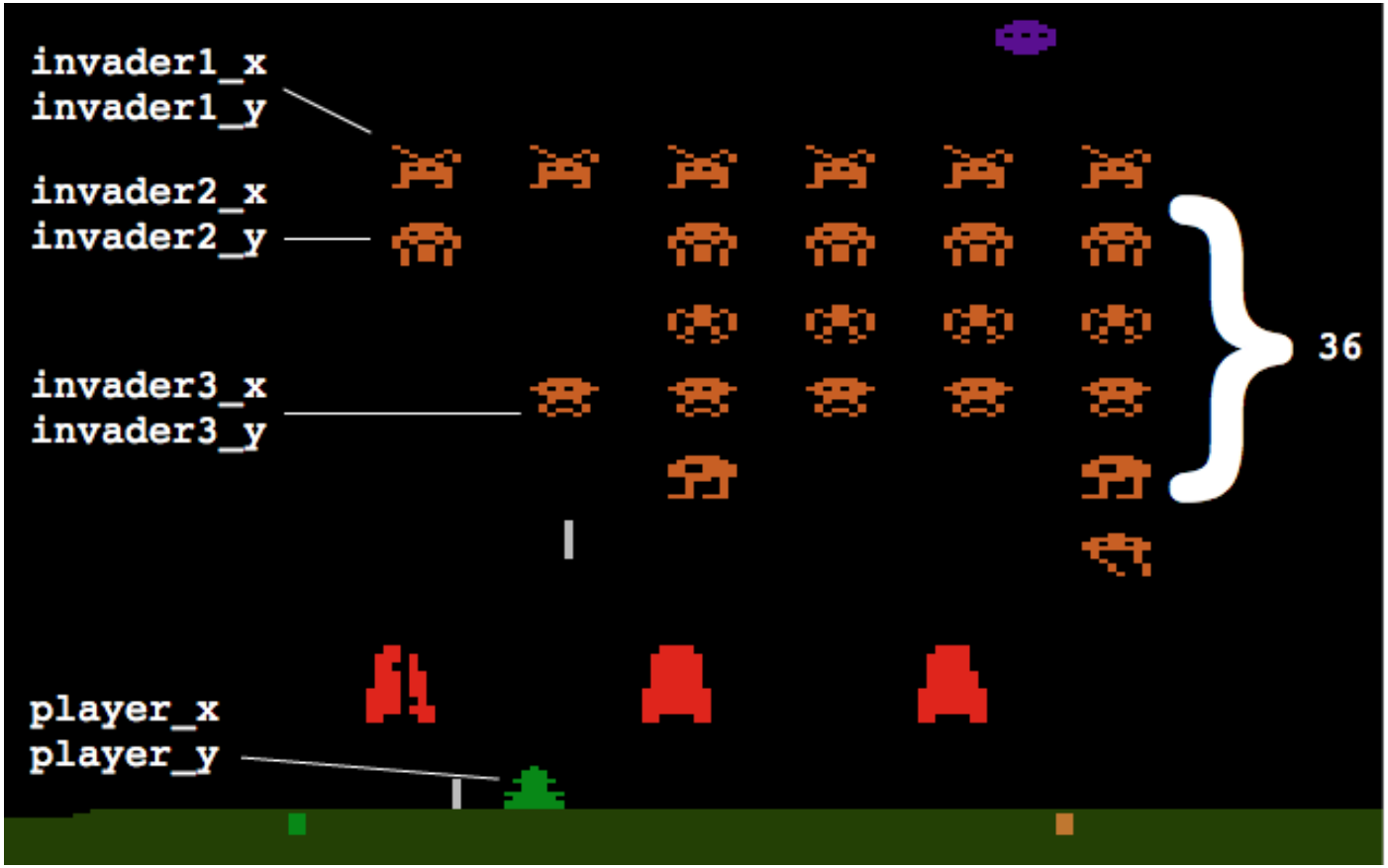
Lists vs. Variables

List	Variable

Variables vs. Lists



Variables vs. Lists



Lists in Python

- You can create a list in Python by using bracket notation. Example:

```
my_list = [1, 2, 3]
```

- The above code will create a new list in Python that holds three integers – 1, 2 and 3 – in that order.
- Think of a list as a "book" that holds a series of sheets of paper (variables)

Lists in Python

- Lists can contain any data type that we have covered so far. Example:

```
my_list = ['Craig', 'John', 'Chris']
```

- Lists can also mix data types. Example:

```
my_list = ['Craig', 5.0, True, 67]
```

- You can print the value of a list using the print() function. Example:

```
print (my_list)
```

List Repetition

- You can use the repetition operation ("*") to ask Python to repeat a list, much like how you would repeat a string. Example:

```
my_list = [1, 2, 3] * 3
print (my_list)

>> [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

List Concatenation

- You can use the concatenation operation ("+") to ask Python to combine lists, much like how you would combine strings. Example:

```
my_list = [1, 2, 3] + [99, 100, 101]
print (my_list)

>> [1, 2, 3, 99, 100, 101]
```

Indexing List Elements

- In a book you can reference a page by its page number
- In a list you can reference an element by its index number
- Indexes start at the number zero.
- Example:

```
my_list = [‘Craig’, ‘John’, ‘Chris’]
print (my_list[0])

>> Craig
```

Invalid indexes

- You will raise an exception if you attempt to access an element outside the range of a list. For example:

```
my_list = ['Craig', 'John', 'Chris’]

print (my_list[4]) # Index doesn’t exist!
```

Changing the value of an item in a list

- Lists are "mutable," which means that they can be changed once they have been created (unlike strings)
- Example:

```
my_list = [1, 2, 3]
print (my_list)
>> [1,2,3]

my_list[0] = 99
print (my_list)
>> [99,2,3]
```

List Mechanics

- List variables are considered "references"
- This means that they "reference" or "point" to a specific region of your computer's memory. This behavior can cause some interesting side effects. For example, the following two list variables refer to the same list in memory.

```
mylist1 = [1,2,3]
mylist2 = mylist1

print (mylist1)
print (mylist2)

>> [1,2,3]
>> [1,2,3]
```

List Mechanics

- This means that you can change one of the lists and the change will be reflected in the other.

```
mylist1 = [1,2,3]
mylist2 = mylist1

mylist1[0] = 999

print (mylist1)
print (mylist2)

>> [999,2,3]
>> [999,2,3]
```

Copying a List

- Python will only create new lists when you use [] syntax to define a list for the first time
- You can take advantage of this behavior to create true copies of your list objects. For example:

```
mylist1 = [1,2,3]
mylist2 = [] + mylist1

mylist1[0] = 999

print (mylist1)
print (mylist2)

>> [999,2,3]
>> [1,2,3]
```

Creating Lists

- You can create an empty list with no elements using the following syntax:

```
mylist = []
```

- Sometimes you want to create a list that contains a certain number of "pre-set" elements. For example, to create a list with 10 elements that are all set to zero you could do the following:

```
mylist = [0] * 10
```

Creating Lists

- You can also create lists using the range() function. For example, to create a list of all even numbers between 0 and 100 you can do the following:

```
even_numbers = list(range(0,100,2))
```

Iterating over a list

Iterating over a list using index values

- There are two main techniques for iterating over a list using index values. They include:
  - Setting up a counter variable outside the list and continually updating the variable as you move to the next position in the list
  - Using the range() function to create a custom range that represents the size of your list

Using a counter variable and a for loop to iterate over a list

- If you set up an accumulator variable outside of your loop you can use it to keep track of where you are in a list. For example:

```
mylist = [1,2,3]
counter = 0

for num in mylist:
     mylist[counter] = mylist[counter] * 2
     counter += 1

print (mylist)

>> [2,4,6]
```

Using the range() function to iterate over a list

- You can also use the range() function to construct a custom range that represents all of the indexes in a list. Example:

```
mylist = [1,2,3]

for counter in range(0,len(mylist)):
    mylist[counter] = mylist[counter] * 2

print (mylist)

>> [2,4,6]
```

Using a "for" loop to iterate through a List

- You can also use a for loop to iterate through a list. When you do this the target variable of your loop assumes each value of each element of the list in order. Example:

```
my_list = [1,2,3]

for number in my_list:
    print (number)

>> 1
>> 2
>> 3
```

Drawbacks to using "for" loops to iterate through a List

- A for loop is a convenient way to sequentially iterate through a list.
- The target variable in a for loop assumes the value of the current item in the list as you iterate.
- However, the target variable isn't very helpful if you want to change the value of an item in a list since it is just a copy of the data that exists in the list. For example:

```
mylist = [1,2,3]

for n in mylist:
      n = n * 5

print (mylist)

>> [1,2,3]
```

Question: What happens if we delete an item while iterating? Does it matter if we use for or for each?

Changing List Items

- In order to change a list item you need to know the index of the item you wish to change. For example:

```
mylist = [1,2,3]

mylist[0] = 999

print (mylist)

>> [999, 2, 3]
```

Programming Challenge

- Given the following list of prices, write a program that modifies the list to include 7% sales tax

```
prices = [1.99, 2.99, 3.99, 4.99, 5.99, 6.99]
```

Programming Challenge: Count the A's

- Given the following list:

```
grades = [90,100,70,45,76,84,93,21,36,99,100]
```

- Write a program that counts the # of A's (scores between 90 and 100)
- Extension: Count the # of B's, C's, D's and F's