Python

Lez go

In the “Interpreter”

```
>>> print(“My code goes next to the carrots”)
My code goes next to the carrots

>>> print(“The output goes in a new line”)
The output goes in a new line
```

Elementary, my dear Python

```
>>> 5 + 4 - 1
8
>>> 6 / 4
1.5
>>> 5 * 4
20
```

```
>>> 14 % 5
4
>>> 6 // 4
1
>>> 3 ** 3
27
```

*In Python 2.7, both / and // are floor divide. (ew)

Assigning a variable

```
>>> my_variable = 14
>>> my_variable
14
>>> foo = 1
>>> my_variable + foo
15
```

Setting the value of my_variable to 14, setting the value of foo to 1.

```
>>> bar = 5
>>> bar = 6
>>> bar / 2
3
>>> bar
6
```

Performing an elementary operation on a variable assigned to a number does not change the value of the variable

A useful operation

```
>>> counter = 3
>>> counter += 1
>>> counter
4
>>> counter -= 1
counter
3
```

+= or -= do two things: They perform an addition/subtraction on the variable, then set the variable to that new value!

How could this be useful?

Strings

```
>>> name = “Steven”
>>> name + “ is a pretty cool guy”
‘Steven is a pretty cool guy’


>>> print(name)
Steven
```

Variables can be assigned to words as well! They are called ‘strings’, and are surrounded by “ or ‘

Some Utilities (Strings)

s[1:]	“All but first”
s[:-1]	“All but last”
s + x	“Add x to the end of s”
x in s	True if chars of x appear in order in s
x not in s	False if chars of x appear in order in s
s[n:k]	char n to char k not including char k from s

Booleans and Logic

```
>>> 5 == 5
True
>>> 5 != 5
False
>>> 5 < 4
False
```

```
>>> True and False
False
>>> True or False
True
>>> not True
False
```

If / Elif / Else

```
>>> if 9 == 5:
...     print(“if case”)
... elif 47 <= 47:
...     print(“else if case”)
... else:
...     print(“else case”)
...
else if case
```

Your friend, the while loop

```
>>> while <this is true>:
...     <do some stuff>
...     <get closer to stopping loop>
```

The ‘while loop’ will repeat until the <this is true> condition becomes false.

A quickie

```
>>> count = 1
>>> python = ‘revolutionary’
>>> while count < 4:
...     python += ‘!’
...     count += 1
...
>>> python
‘revolutionary!!!’
```

Defining Functions

```
>>> def my_func(x, y):
...         return x * y
...
>>> my_func(5, 6)
30
```

It’s easy to write functions in Python! “def” followed by your function’s name, followed by (your variables): will get you started!

A note on Indentation

Indents (4 spaces) are important in Python when they are preceding statements.

```
>>> def exclaimer(word):
...     for i in range(1, 3):
...         k = 1
...         while k < 4:
...             word = word + ‘!’
...             k = k + 1
...     return word
```

Think of it like nesting in Snap!

If / Else and Indentation

```
>>> cookie = “delicious”
>>> if len(cookie) == 9:
...     print(exclaimer(cookie))
... else:
...     print(‘not 9 characters’)
...
delicious!!!!!!
```

To the Prompts!

>>>

Greetings!

```
def greet(name):
    “””Give a greeting.
    >>> greet(“Johnny”)
    Hello Johnny
    ”””
```

Greetings!

```
def greet(name):
  print(“Hello ” + name)
```

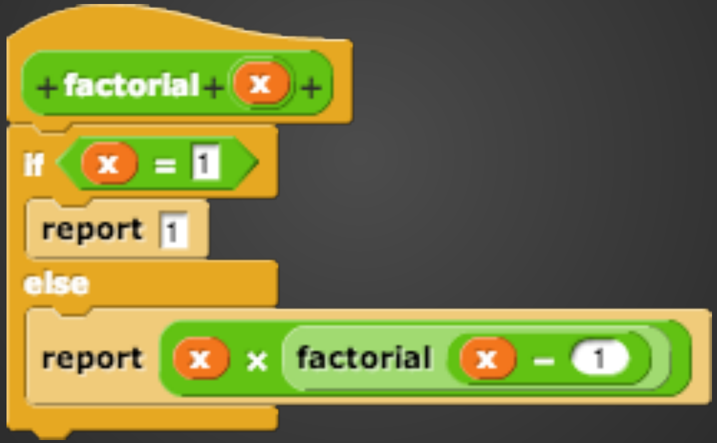
Factorial (again…):

```
def factorial(x):
   “””Return the factorial of x.”””
```

Factorial (again...):

```
def factorial(x):
  if x == 1:
    return 1
  else:
    return x * factorial(x - 1)
>>> factorial(5)
120
```

Factorial (Snap!):



```
+ factorial + x +
if x = 1
    report 1
else
    report x × factorial x − 1
```

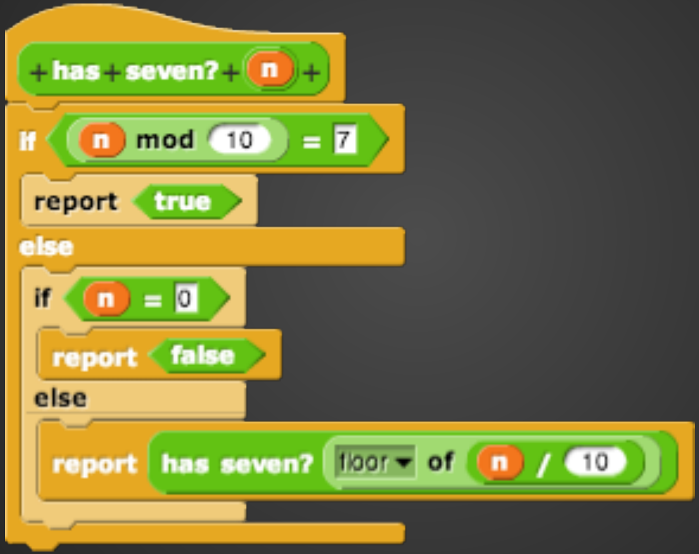
Has seven?

```
def has_seven(n):
    “““Given a number n, return whether any of its digits
    is a 7.
    (hint: floor division and modulo might be helpful)
    ”””
>>> has_seven(45)
False
>>> has_seven(20178)
True
```

Has seven?

```
def has_seven(n):
    if n % 10 == 7:
        return True
    elif n == 0:
        return False
    else:
        return has_seven(n // 10)
>>> has_seven(453)
False
>>> has_seven(979)
True
```

Has seven (Snap!)?



```
+has+seven?+(n)+
if ((n) mod (10)) = 7
    report true
else
    if (n) = 0
        report false
    else
        report has seven? (floor of ((n) / (10)))
```

Every other character in string

```
def every_other(string):
    “””Given a string, return a new string with
    only every other character of the original.”””
```

Every other character in string

```
def every_other(string):
    output_string = “”
    for i in range(len(string)):
        if i % 2 == 0:
            output_string = output_string + string[i]
    return output_string
```

Notice that “i” here represents the *index* in the string

Factorion

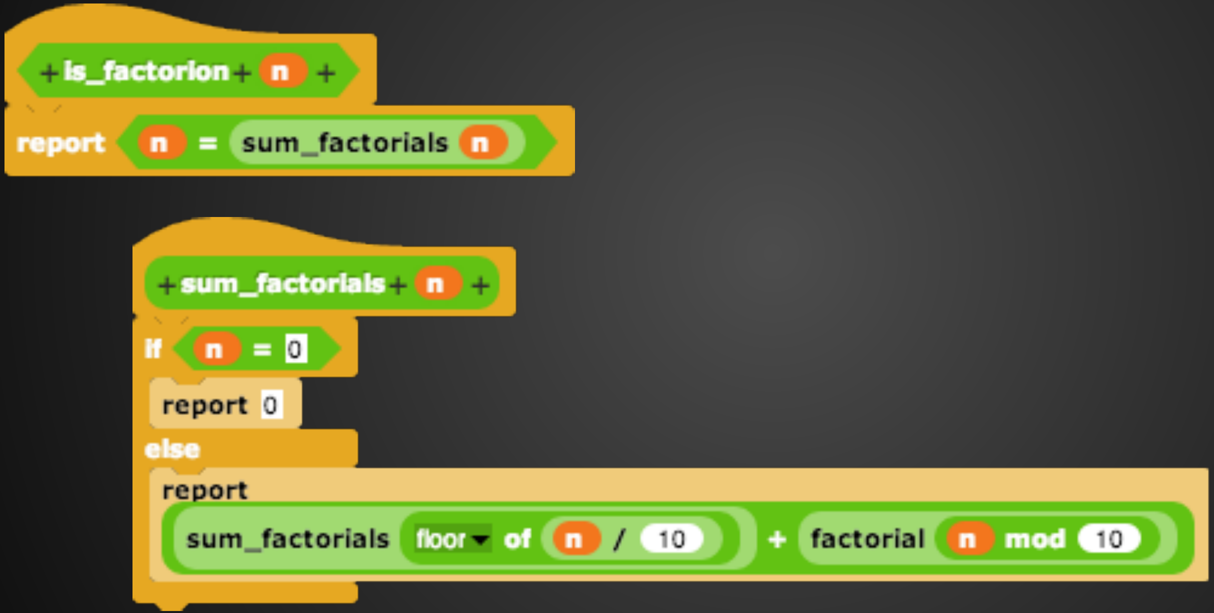
```
def is_factorion(n):
    “””Return whether the sum of the factorials of
    n’s digits add up to n.
    (hint: floor division and modulo might be
    helpful) ”””
    # The ‘#’ is used to create one-line comments.
    # You can assume factorial(n) is already written.
```

Factorion - Recursive

```
def is_factorion(n):
   return n == calc_factorion(n)

def calc_factorion(n):
   if n == 0:
      return 0
   return calc_factorion(n//10) + factorial(n%10)
```

Factorion - Recursive



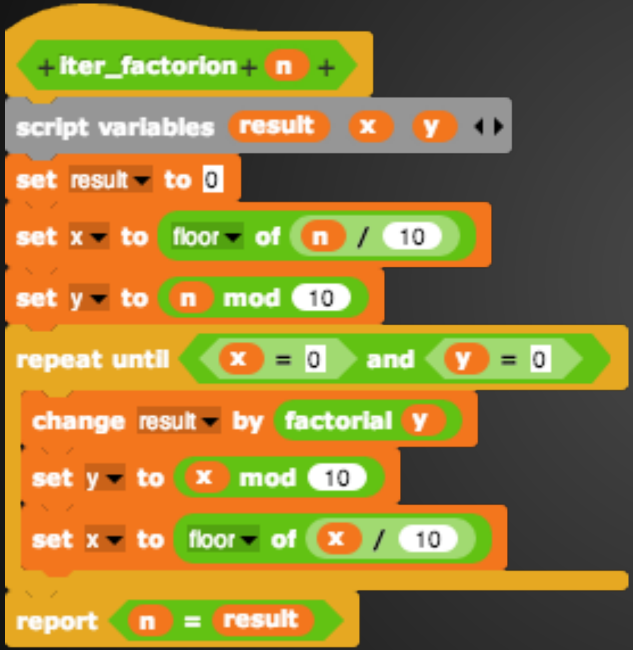
```
+ is_factorion + n +
report ( n = sum_factorials n )

+ sum_factorials + n +
if ( n = 0 )
    report 0
else
    report
    ( sum_factorials floor of ( n / 10 ) + factorial ( n mod 10 ) )
```

Factorion - Iterative

```python
def iter_factorion(n):
    result = 0
    x = n // 10
    y = n % 10
    while not(x==0 and y==0):
        result += factorial(y)
        y = x % 10
        x = x // 10
    return n == result
```

Factorion - Iterative



```
+ Iter_factorion + n +
script variables result x y ◂▸
set result to 0
set x to floor of (n / 10)
set y to (n mod 10)
repeat until <<(x) = 0> and <(y) = 0>>
    change result by factorial y
    set y to (x mod 10)
    set x to floor of (x / 10)
report <(n) = (result)>
```