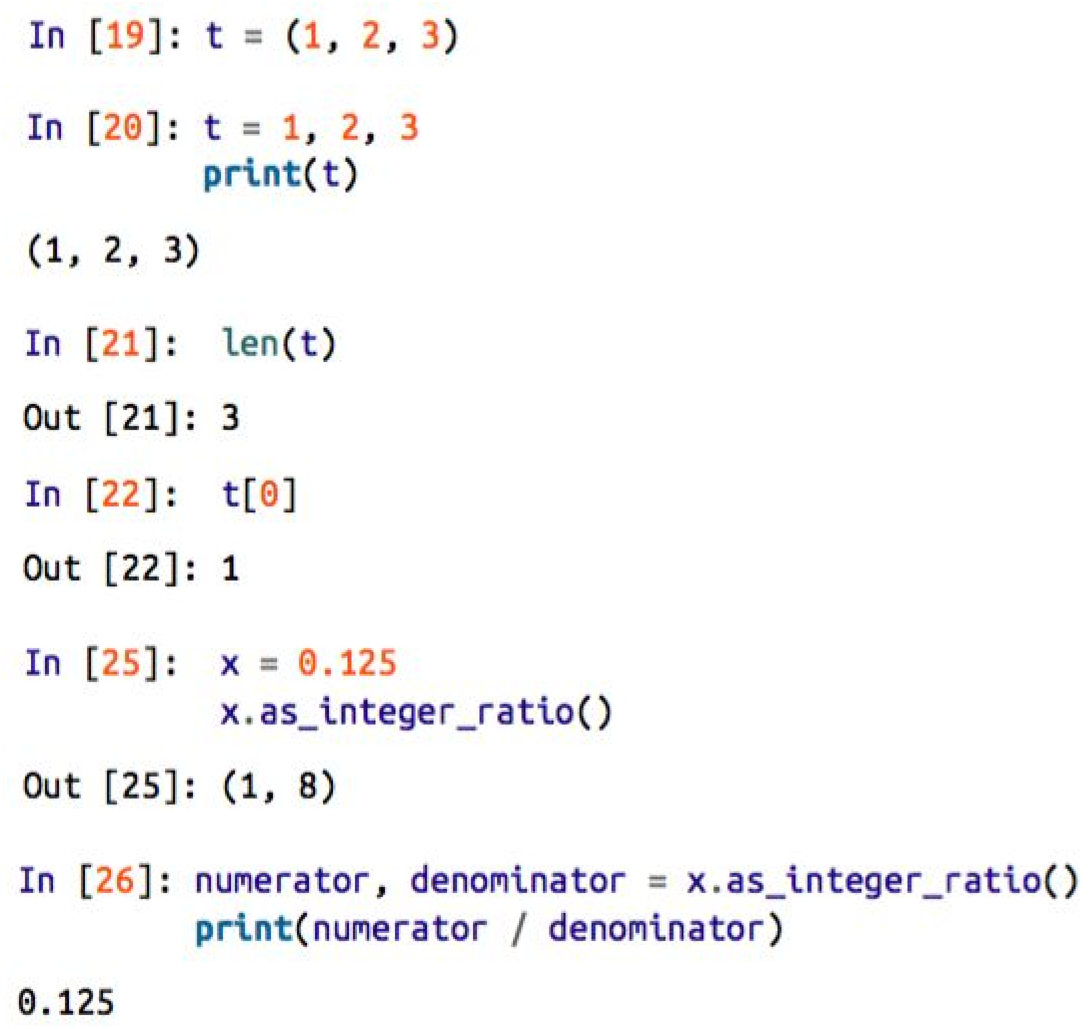
Python III

Built-in Data Structures, Control Flow

Tuples

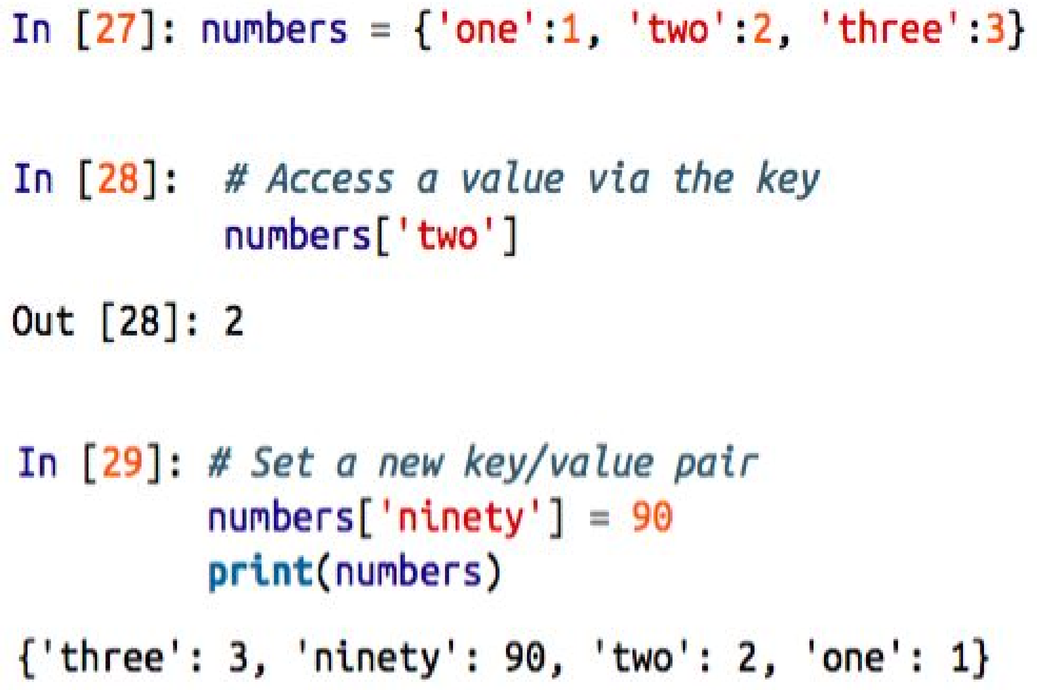
- Create a tuble with parentheses
- They are immutable
- Get length with len()
- Often used when functions return multiple values



```
In [19]: t = (1, 2, 3)

In [20]: t = 1, 2, 3
         print(t)

(1, 2, 3)

In [21]:  len(t)

Out [21]: 3

In [22]:  t[0]

Out [22]: 1

In [25]:  x = 0.125
          x.as_integer_ratio()

Out [25]: (1, 8)

In [26]: numerator, denominator = x.as_integer_ratio()
         print(numerator / denominator)

0.125
```

Dictionaries

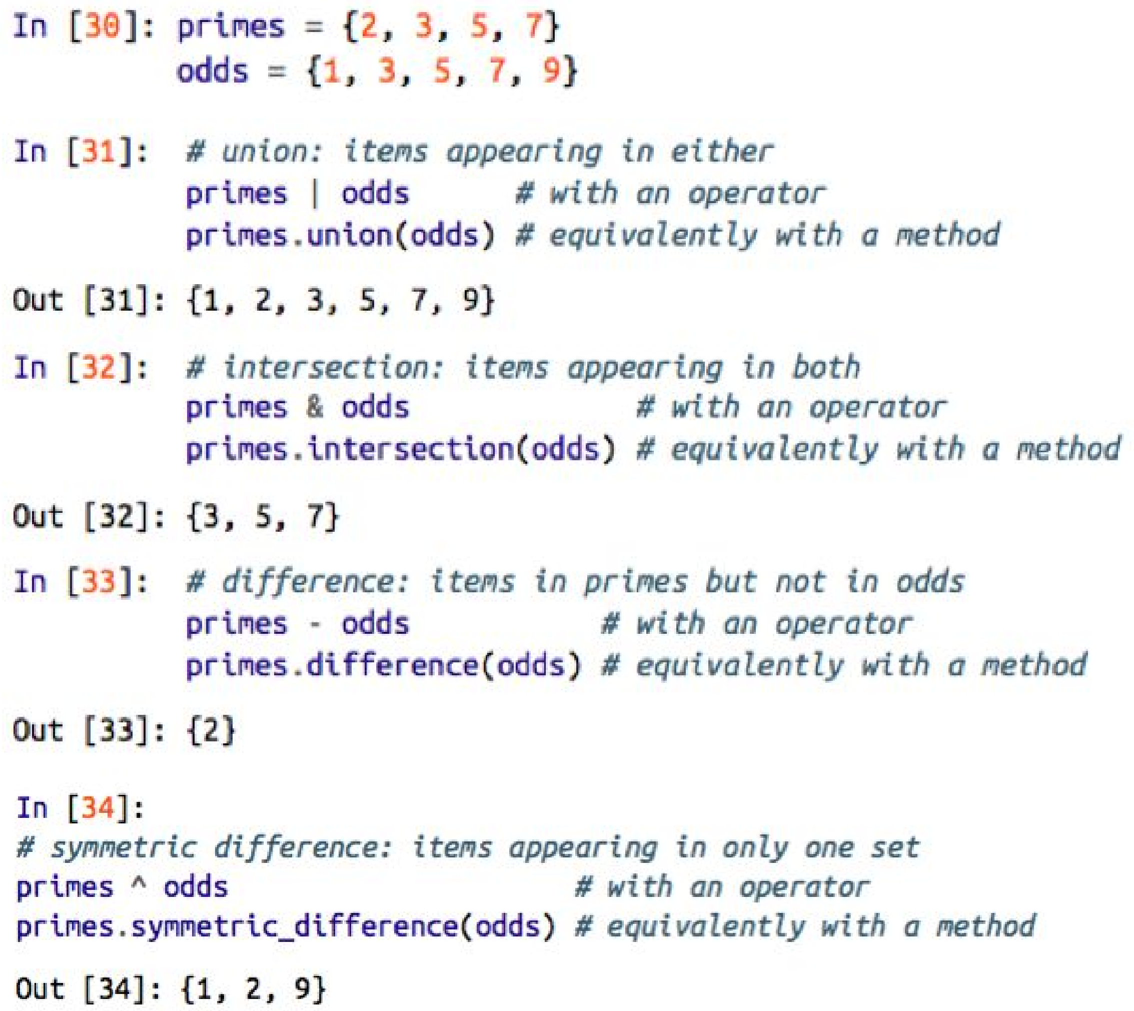
- An unordered collection of key/value pairs
- Syntax for using them is on the right



```
In [27]: numbers = {'one':1, 'two':2, 'three':3}

In [28]:  # Access a value via the key
          numbers['two']

Out [28]: 2

In [29]: # Set a new key/value pair
         numbers['ninety'] = 90
         print(numbers)

{'three': 3, 'ninety': 90, 'two': 2, 'one': 1}
```

Sets

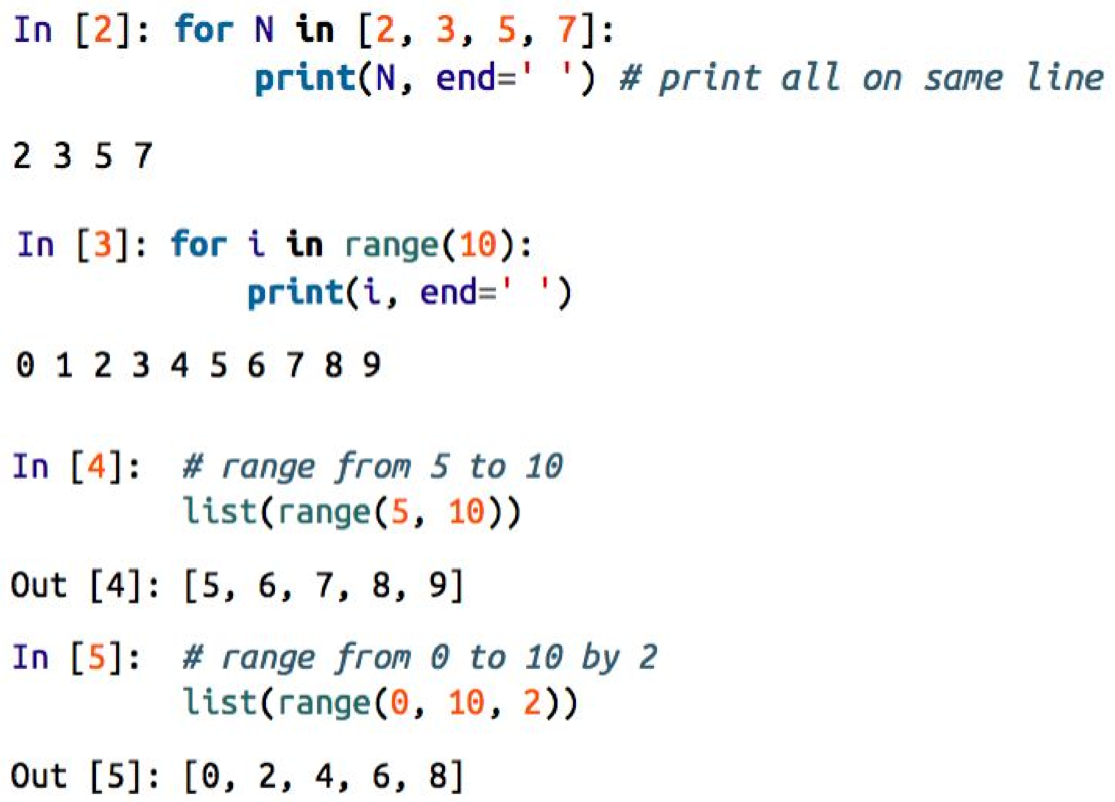
- An unordered collection of unique items
- Some set operations are on the right



```
In [30]: primes = {2, 3, 5, 7}
         odds = {1, 3, 5, 7, 9}

In [31]:  # union: items appearing in either
          primes | odds        # with an operator
          primes.union(odds) # equivalently with a method

Out [31]: {1, 2, 3, 5, 7, 9}

In [32]:  # intersection: items appearing in both
          primes & odds               # with an operator
          primes.intersection(odds) # equivalently with a method

Out [32]: {3, 5, 7}

In [33]:  # difference: items in primes but not in odds
          primes - odds             # with an operator
          primes.difference(odds) # equivalently with a method

Out [33]: {2}

In [34]:
# symmetric difference: items appearing in only one set
primes ^ odds                       # with an operator
primes.symmetric_difference(odds) # equivalently with a method

Out [34]: {1, 2, 9}
```

Control flow: conditional statements

if, else, elif (a combination of else and if)



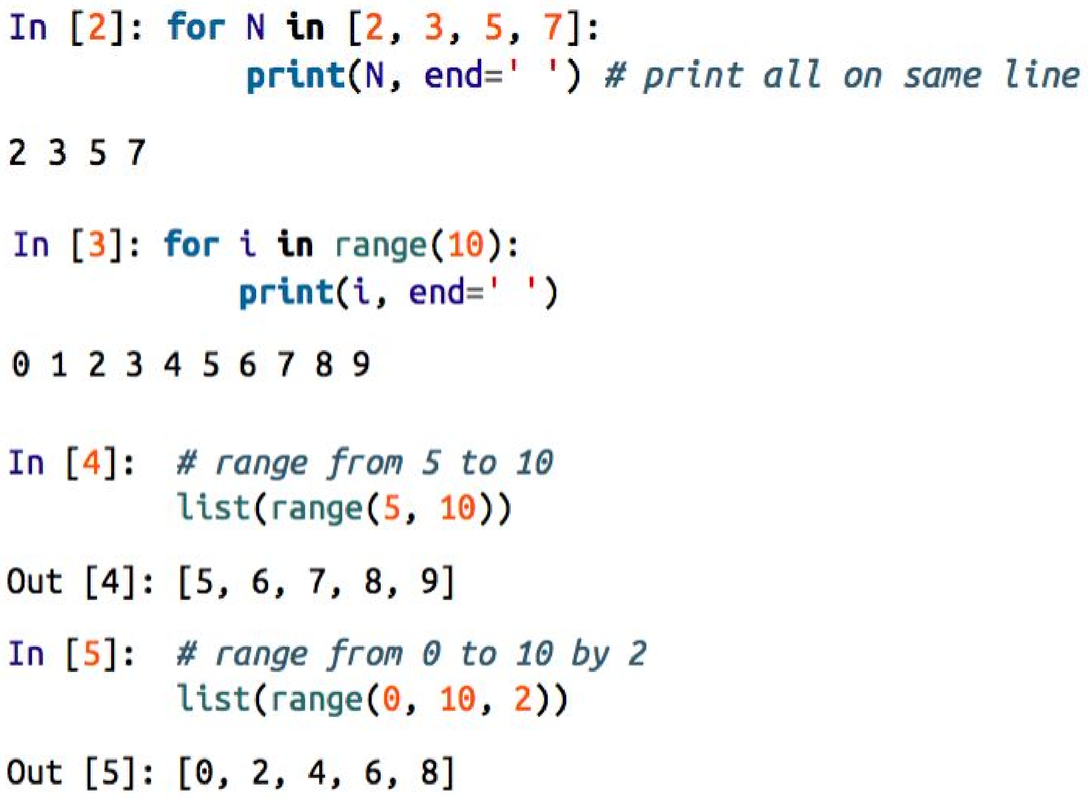
```
In [2]: for N in [2, 3, 5, 7]:
            print(N, end=' ') # print all on same line

2 3 5 7

In [3]: for i in range(10):
            print(i, end=' ')

0 1 2 3 4 5 6 7 8 9

In [4]:   # range from 5 to 10
          list(range(5, 10))

Out [4]: [5, 6, 7, 8, 9]

In [5]:   # range from 0 to 10 by 2
          list(range(0, 10, 2))

Out [5]: [0, 2, 4, 6, 8]
```

For loops

Accomplish repetitive tasks

Loop over an iterator using the `in` keyword

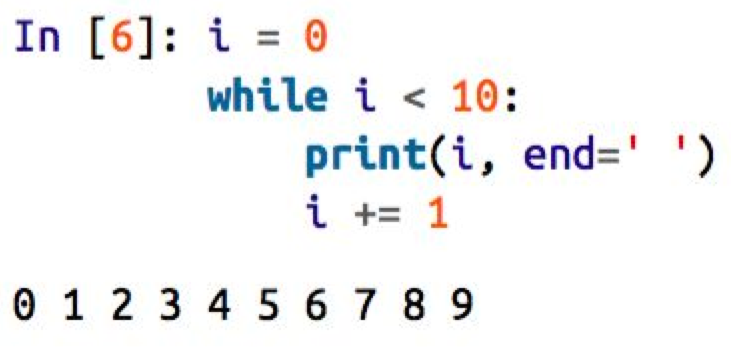
`range(x, y, z)` produces a list with values numbering from x to y-1 with a step of z



```
In [2]: for N in [2, 3, 5, 7]:
            print(N, end=' ') # print all on same line

2 3 5 7

In [3]: for i in range(10):
            print(i, end=' ')

0 1 2 3 4 5 6 7 8 9

In [4]:  # range from 5 to 10
         list(range(5, 10))

Out [4]: [5, 6, 7, 8, 9]

In [5]:  # range from 0 to 10 by 2
         list(range(0, 10, 2))

Out [5]: [0, 2, 4, 6, 8]
```

While loops

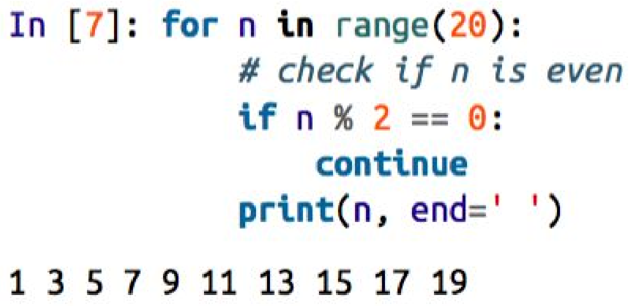
The loop repeats until the condition is false



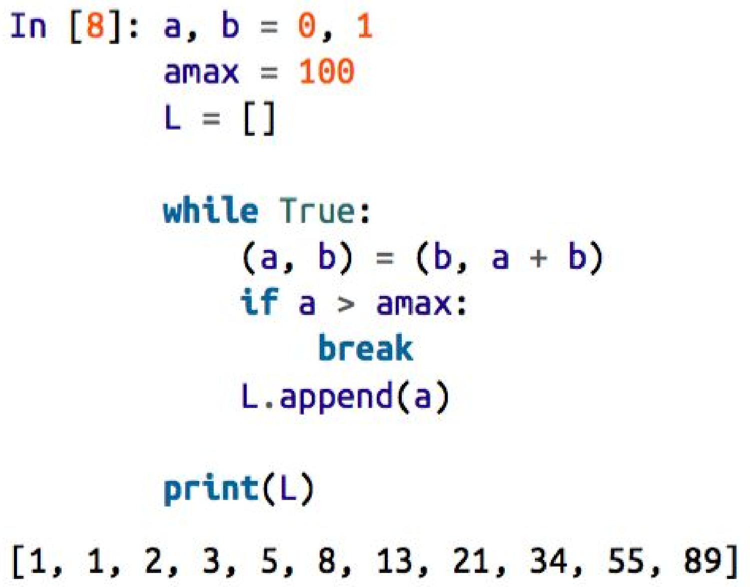
```
In [6]: i = 0
        while i < 10:
            print(i, end=' ')
            i += 1

0 1 2 3 4 5 6 7 8 9
```

Break and continue

- `break` breaks out of the loop
- `continue` skips the rest of the code in the block and goes to the next iteration of the loop



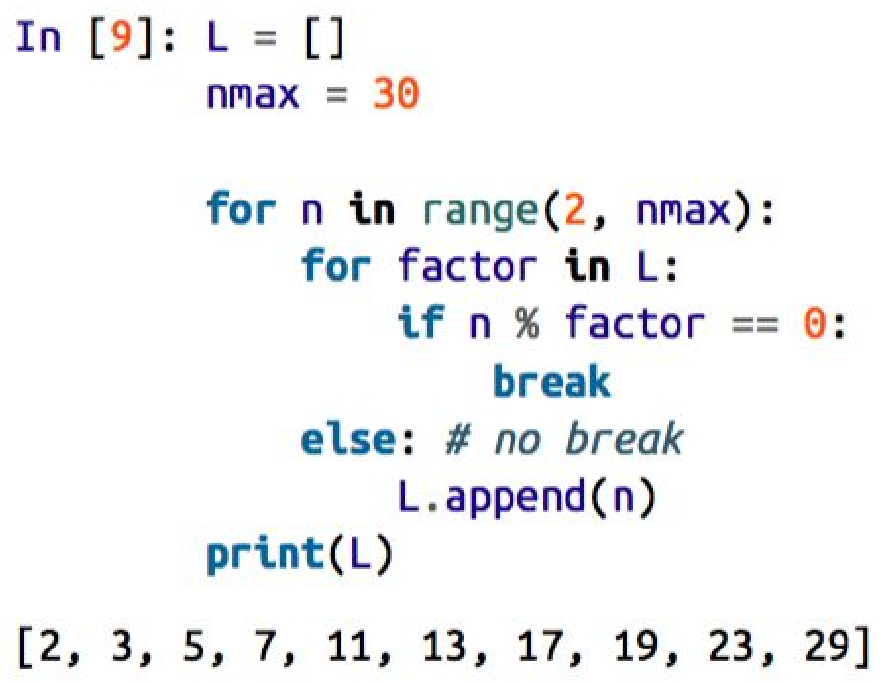
```
In [7]: for n in range(20):
            # check if n is even
            if n % 2 == 0:
                continue
            print(n, end=' ')

1 3 5 7 9 11 13 15 17 19
```



```
In [8]: a, b = 0, 1
        amax = 100
        L = []

        while True:
            (a, b) = (b, a + b)
            if a > amax:
                break
            L.append(a)

        print(L)

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Loops with else block

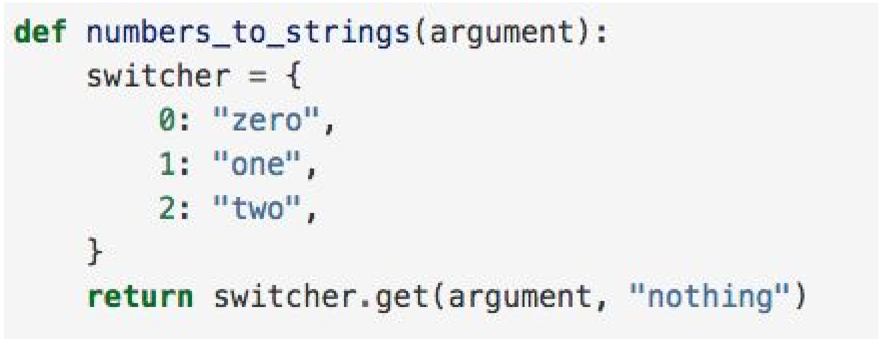
Code in the else block runs only if break is not triggered



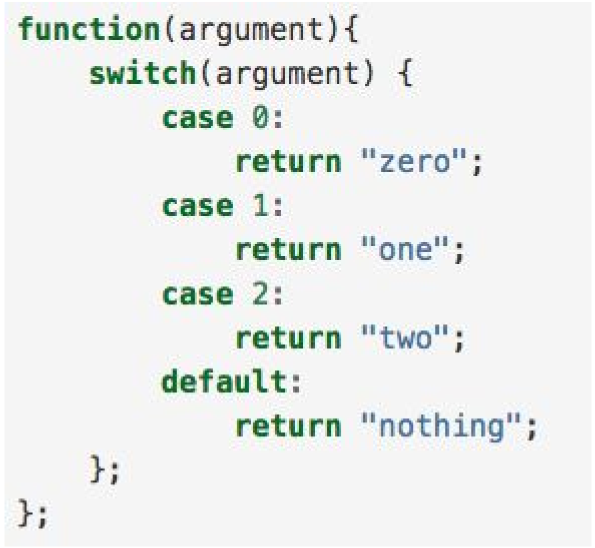
```
In [9]: L = []
        nmax = 30

        for n in range(2, nmax):
            for factor in L:
                if n % factor == 0:
                    break
            else: # no break
                L.append(n)
        print(L)

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

An alternative to switch

- Python doesn’t have a switch statement
- Use a dictionary instead: store what you want to switch on as keys, and the values you want returned as values
- You can also store lambda functions as values (more next week)



```python
def numbers_to_strings(argument):
    switcher = {
        0: "zero",
        1: "one",
        2: "two",
    }
    return switcher.get(argument, "nothing")
```



```javascript
function(argument){
    switch(argument) {
        case 0:
            return "zero";
        case 1:
            return "one";
        case 2:
            return "two";
        default:
            return "nothing";
    };
};
```

A JavaScript equivalent