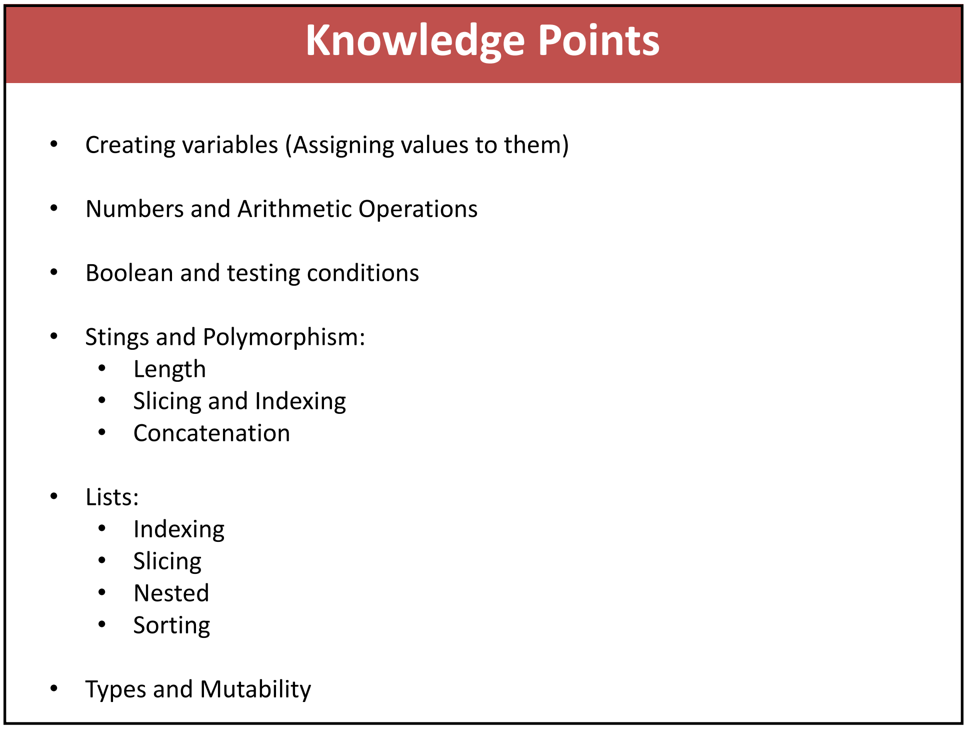
Module 2: Numbers, Strings, and Lists

1

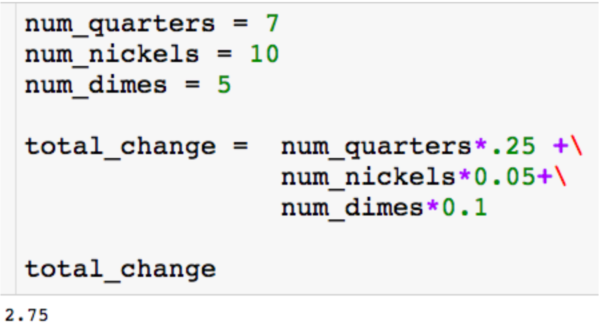


Knowledge Points

- Creating variables (Assigning values to them)
- Numbers and Arithmetic Operations
- Boolean and testing conditions
- Stings and Polymorphism:
  - Length
  - Slicing and Indexing
  - Concatenation
- Lists:
  - Indexing
  - Slicing
  - Nested
  - Sorting
- Types and Mutability

2

Using Variables



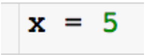
```
num_quarters = 7
num_nickels = 10
num_dimes = 5

total_change =  num_quarters*.25 +\
                num_nickels*0.05+\
                num_dimes*0.1

total_change
```

```
2.75
```

3

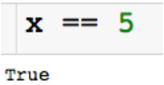
Conditional Tests

- Sets the variables x equal to 5.



```
x = 5
```

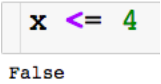
- Asks if x is equal to 5. Returns boolean.



```
x == 5
```

```
True
```

- Asks if x is less than or equal to 4. Returns boolean.



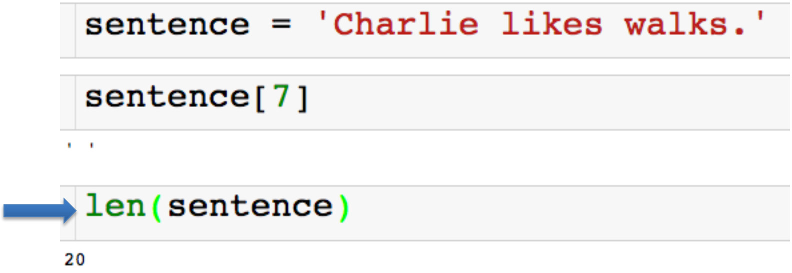
```
x <= 4
```

```
False
```

4

Slicing Strings

We can access the characters of the string through their **index**



```
sentence = 'Charlie likes walks.'
```

```
sentence[7]
```

```
' '
```

```
len(sentence)
```

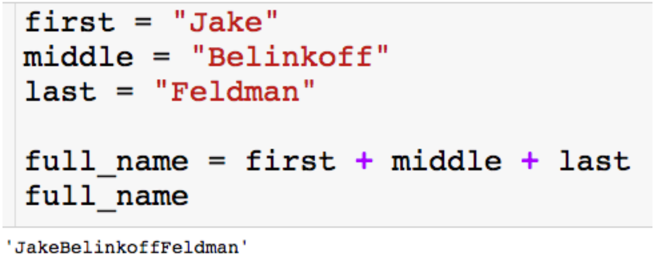
```
20
```

Returns the number of characters in the string

5

String Concatenation

- I can combine strings using the + operator.
- So the + operator between two numbers add them and the + operator between two strings concatenates them! This is called **polymorphism.**



```
first = "Jake"
middle = "Belinkoff"
last = "Feldman"

full_name = first + middle + last
full_name
```

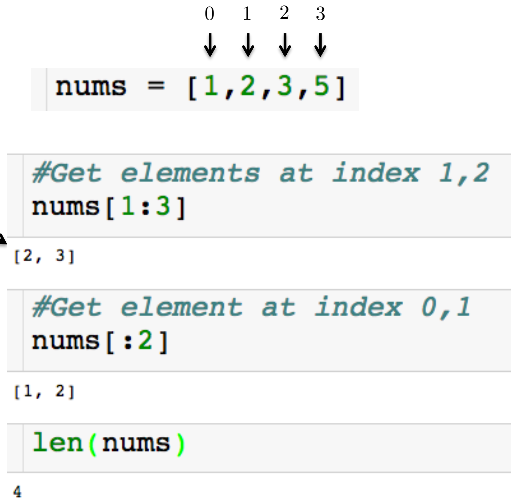
```
'JakeBelinkoffFeldman'
```

- If we want a space, we have to say so.

6

Slicing Lists

- Slicing for lists I also very similar to strings



```
nums = [1,2,3,5]
```

Returns lists

```
#Get elements at index 1,2
nums[1:3]
```

```
[2, 3]
```

```
#Get element at index 0,1
nums[:2]
```

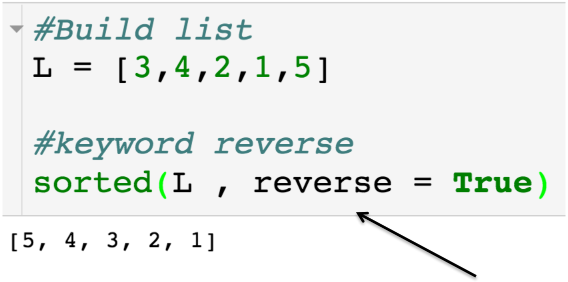
```
[1, 2]
```

```
len(nums)
```

```
4
```

Sorting Lists

- We can sort lists with the built-in sorted() function.



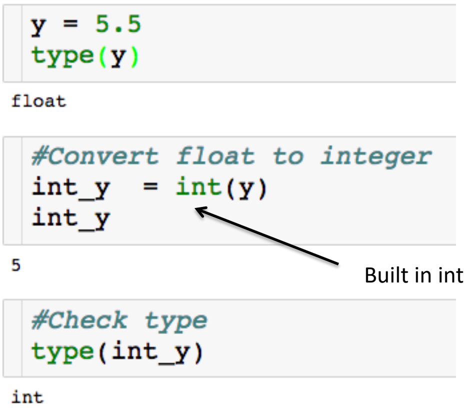
```
#Build list
L = [3,4,2,1,5]

#keyword reverse
sorted(L , reverse = True)
```

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

- Sort list descending
- Default is reverse = False

Next session we will see how to sort L “inplace”.

Converting Types



```
y = 5.5
type(y)
```

```
float
```

```
#Convert float to integer
int_y  = int(y)
int_y
```

```
5
```

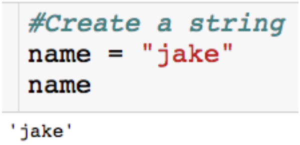
Built in int() function

```
#Check type
type(int_y)
```

```
int
```

- int() is one way to perform a floor operation

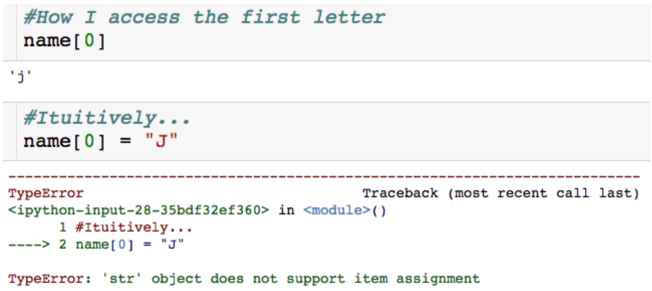
Example of Immutability



```
#Create a string
name = "jake"
name
```

```
'jake'
```

Let’s say I want to change the first letter of name to a “J”



```
#How I access the first letter
name[0]
```

```
'j'
```

```
#Ituitively...
name[0] = "J"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-35bdf32ef360> in <module>()
      1 #Ituitively...
----> 2 name[0] = "J"

TypeError: 'str' object does not support item assignment
```

Can’t change name once it is created!

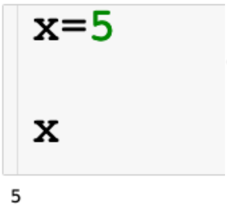
Intro to Python Objects – Part 1

python™

11

Creating Variables

Let's create our first variables



```
x=5

x
```

```
5
```

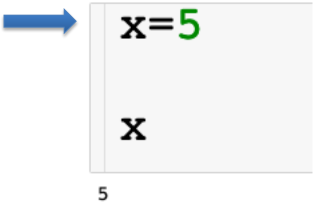
Code cell

The code executes from top to bottom

12

Creating Variables

Let’s create our first variables



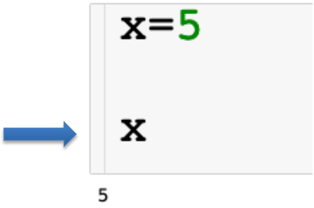
```
x=5

x
```

```
5
```

$x = 5$

Creating Variables

Let’s create our first variables



```
x=5

x
```

```
5
```

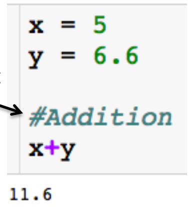
$x = 5$

Python Objects

- Variables are simply names that are used to keep track of information.
  - Variables are created when they are first assigned a value.
  - Variables must be assigned before they can be used.
- Variables will take the form of Python objects. We will use 3 different objects:
  - **Numbers**: integers, real number, etc …
  - **Strings**: ordered sequences of characters
  - **Lists**: ordered collection of objects
- Python objects are **dynamically typed**, meaning you don't have to declare the type of the variable upon creation.

15

Arithmetic Operations



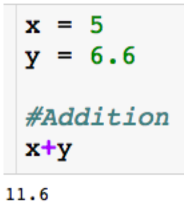
```
x = 5
y = 6.6

#Addition
x+y
```

```
11.6
```

# = comment

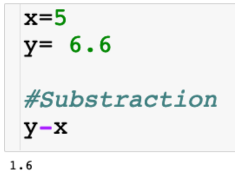
16

Arithmetic Operations



```
x = 5
y = 6.6

#Addition
x+y
```

```
11.6
```

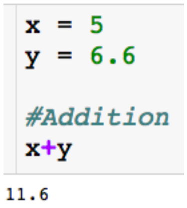


```
x=5
y= 6.6

#Substraction
y-x
```

```
1.6
```

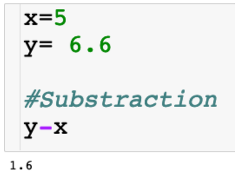
17

Arithmetic Operations



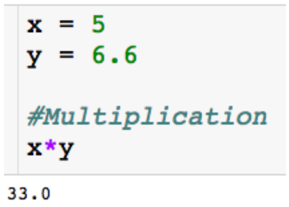
```
x = 5
y = 6.6

#Addition
x+y
```

```
11.6
```



```
x=5
y= 6.6

#Substraction
y-x
```

```
1.6
```

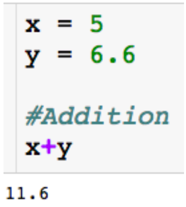


```
x = 5
y = 6.6

#Multiplication
x*y
```

```
33.0
```

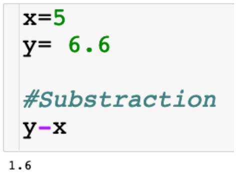
18

Arithmetic Operations



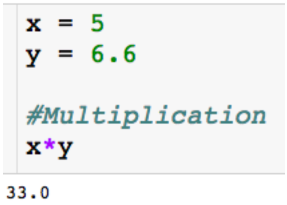
```
x = 5
y = 6.6

#Addition
x+y
```

```
11.6
```



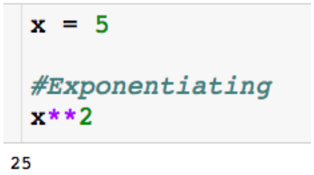
```
x=5
y= 6.6

#Substraction
y-x
```

```
1.6
```



```
x = 5
y = 6.6

#Multiplication
x*y
```

```
33.0
```



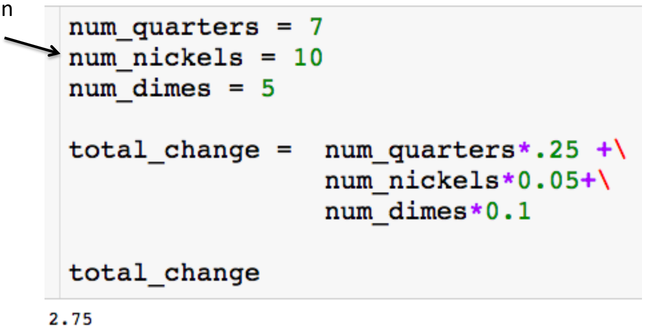
```
x = 5

#Exponentiating
x**2
```

```
25
```

19

Using Variables

Use description variable name



```
num_quarters = 7
num_nickels = 10
num_dimes = 5

total_change =  num_quarters*.25 +\
                num_nickels*0.05+\
                num_dimes*0.1

total_change
```

```
2.75
```

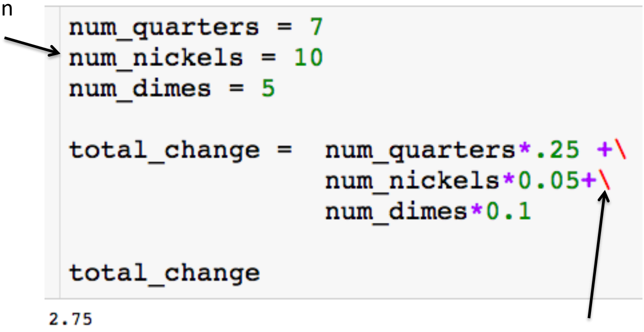
Rule for creating variable names:

- Be descriptive and separate words with underscore
- No spaces
- No punctuation other than underscore

20

Using Variables

Use description variable name



```
num_quarters = 7
num_nickels = 10
num_dimes = 5

total_change =  num_quarters*.25 +\
                num_nickels*0.05+\
                num_dimes*0.1

total_change
```

```
2.75
```

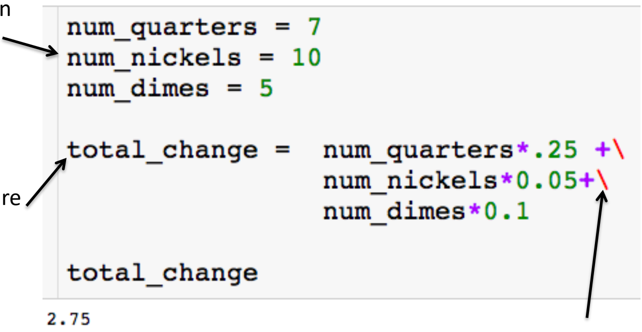
The backslash lets you continue your block of code on the next line.

21

Using Variables

Use description variable name

I can create variables that are a function of other variables



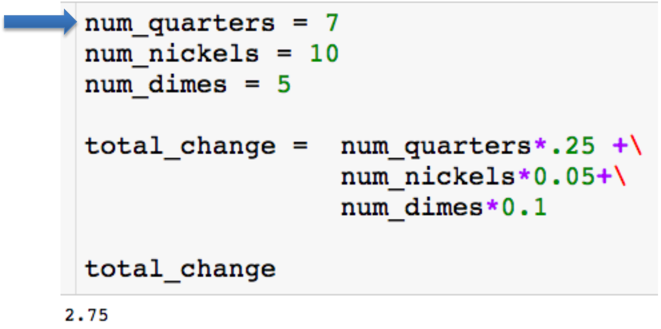
```
num_quarters = 7
num_nickels = 10
num_dimes = 5

total_change =  num_quarters*.25 +\
                num_nickels*0.05+\
                num_dimes*0.1

total_change
```

```
2.75
```

The backslash lets you continue your block of code on the next line.

22

Using Variables



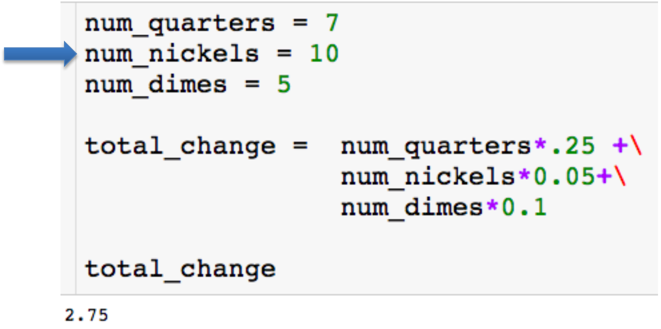
```
num_quarters = 7
num_nickels = 10
num_dimes = 5

total_change =  num_quarters*.25 +\
                num_nickels*0.05+\
                num_dimes*0.1

total_change
```

```
2.75
```

num_quarters = 7

23

Using Variables



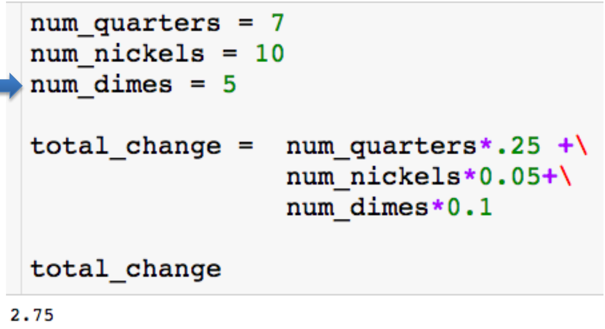
```
num_quarters = 7
num_nickels = 10
num_dimes = 5

total_change =  num_quarters*.25 +\
                num_nickels*0.05+\
                num_dimes*0.1

total_change
```

```
2.75
```

num_quarters = 7
num_nickels = 10

24

Using Variables

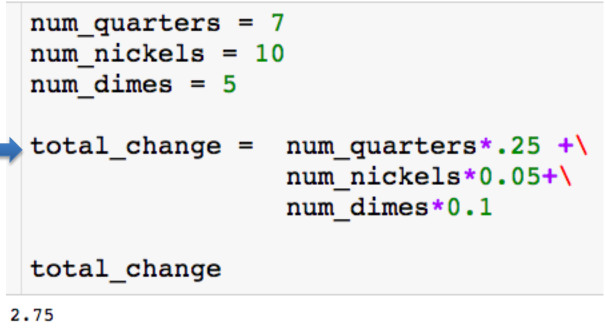


```
num_quarters = 7
num_nickels = 10
num_dimes = 5

total_change =  num_quarters*.25 +\
                num_nickels*0.05+\
                num_dimes*0.1

total_change
```

```
2.75
```

num_quarters = 7
num_nickels = 10
num_nickels = 5

Using Variables

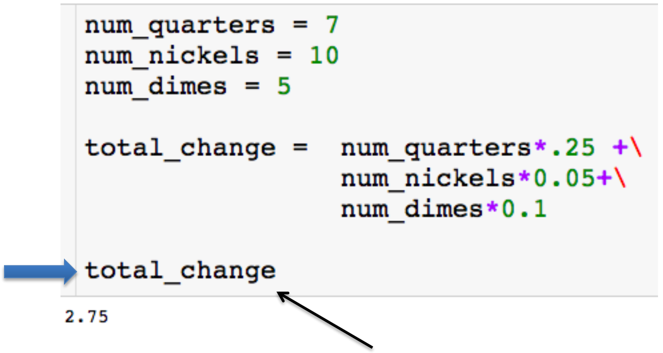


```
num_quarters = 7
num_nickels = 10
num_dimes = 5

total_change =  num_quarters*.25 +\
                num_nickels*0.05+\
                num_dimes*0.1

total_change
```

```
2.75
```

num_quarters = 7
num_nickels = 10
num_nickels = 5
total_change = 2.75

Using Variables



```
num_quarters = 7
num_nickels = 10
num_dimes = 5

total_change =  num_quarters*.25 +\
                num_nickels*0.05+\
                num_dimes*0.1

total_change
```

```
2.75
```

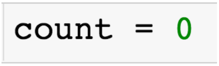
This just prints the value stored in the variable so we can see it.

num_quarters = 7
num_nickels = 10
num_nickels = 5
total_change = 2.75

27

Using Variables

You will often find yourself updating variables:



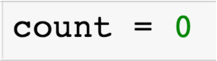
```
count = 0
```

Some other code executes…want to add 1 to count

28

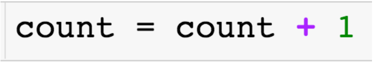
Using Variables

You will often find yourself updating variables:



```
count = 0
```

Some other code executes…want to add 1 to count

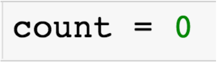


```
count = count + 1
```

29

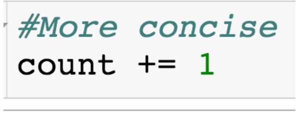
Using Variables

You will often find yourself updating variables:



```
count = 0
```

Some other code executes…want to add 1 to count



```
#More concise
count += 1
```

30

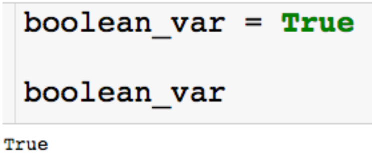
Booleans

- The Boolean type can be viewed as numeric in nature because its values (True and False) are just customized versions of the integers 1 and 0.
- The True and False behave in the same way as 1 and 0, they just make the code more readable.
- Booleans are the type returned when we check if a condition is true

31

Booleans

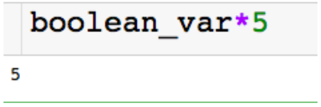
- Creating boolean variable:



```
boolean_var = True

boolean_var
```

```
True
```

- Note that the boolean does behave exactly like a 1:



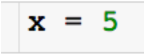
```
boolean_var*5
```

```
5
```

32

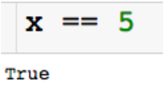
Conditional Tests

- Sets the variables x equal to 5.



```
x = 5
```

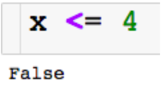
- Asks if x is equal to 5. Returns boolean.



```
x == 5
```

```
True
```

- Asks if x is less than or equal to 4. Returns boolean.



```
x <= 4
```

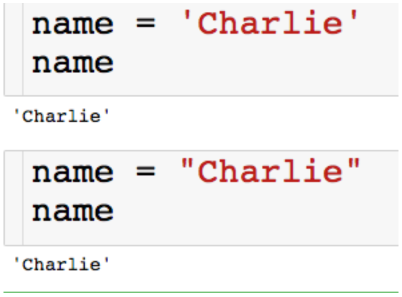
```
False
```

Strings

- Python strings are an ordered collection of characters (usually these characters will be letters and numbers) used to represent text.
- String are created by placing single or double quotation marks around a sequence of characters.
- Strings support the following operations
  - concatenation (combining strings)
  - slicing (extracting sections)
  - Indexing (fetching by offset)
  - the list goes on ….

Strings

Let’s create our first strings



```
name = 'Charlie'
name
```

```
'Charlie'
```

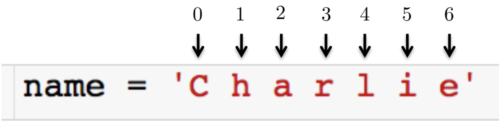
```
name = "Charlie"
name
```

```
'Charlie'
```

- You can create a string with either single or double quotes.
- There is a left to right ordering that we will explore on the next slide

Indexing Strings

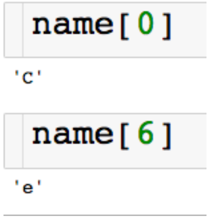
We can access the characters of the string through their **index**



```
name = 'C h a r l i e'
```

(pretend there aren’t spaces between the letters)

Slicing single characters through index:



```
name[0]
```

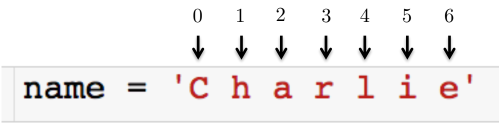
```
'C'
```

```
name[6]
```

```
'e'
```

Slicing Strings

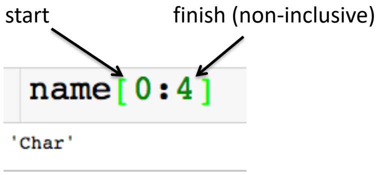
We can access the characters of the string through their **index**



```
name = 'C h a r l i e'
```

(pretend there aren't spaces between the letters)

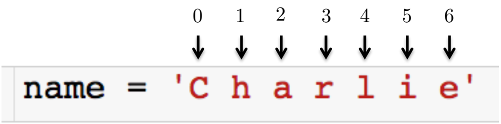
Slicing contiguous characters:



```
name[0:4]
'Char'
```

Slicing Strings

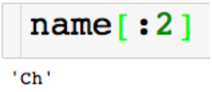
We can access the characters of the string through their **index**



```
name = 'C h a r l i e'
```

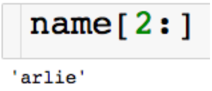
(pretend there aren't spaces between the letters)

Slicing contiguous characters:



```
name[:2]
'Ch'
```

If start index is left blank defaults to 0

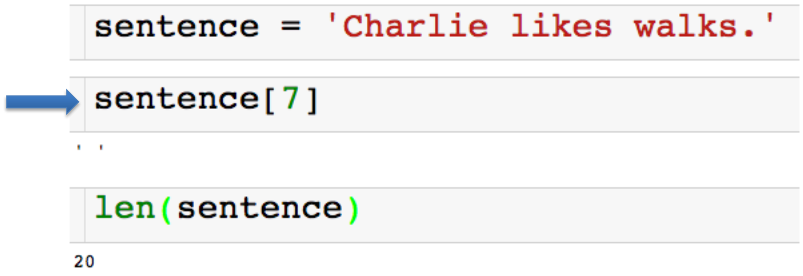


```
name[2:]
'arlie'
```

If end index is left blank defaults to end of the string

Slicing Strings

We can access the characters of the string through their **index**



```
sentence = 'Charlie likes walks.'
```

```
sentence[7]
```

```
' '
```

```
len(sentence)
```

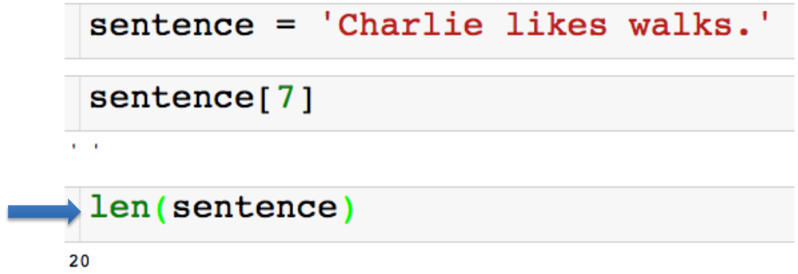
```
20
```

Spaces and punctuation count in the indexing of a string!

39

Slicing Strings

We can access the characters of the string through their **index**



```
sentence = 'Charlie likes walks.'
```

```
sentence[7]
```

```
' '
```

```
len(sentence)
```

```
20
```

Returns the number of characters in the string

40

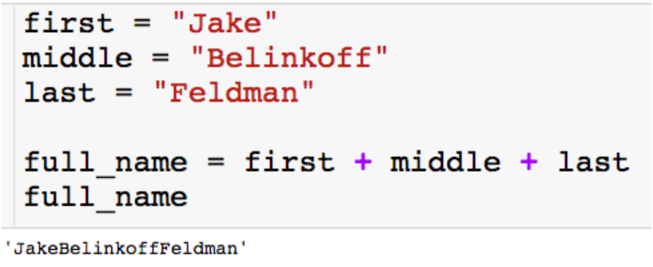
String Concatenation

- I can combine strings using the + operator.
- So the + operator between two numbers add them and the + operator between two strings concatenates them! This is called **polymorphism.**

41

String Concatenation

- I can combine strings using the + operator.
- So the + operator between two numbers add them and the + operator between two strings concatenates them! This is called **polymorphism.**



```
first = "Jake"
middle = "Belinkoff"
last = "Feldman"

full_name = first + middle + last
full_name
```

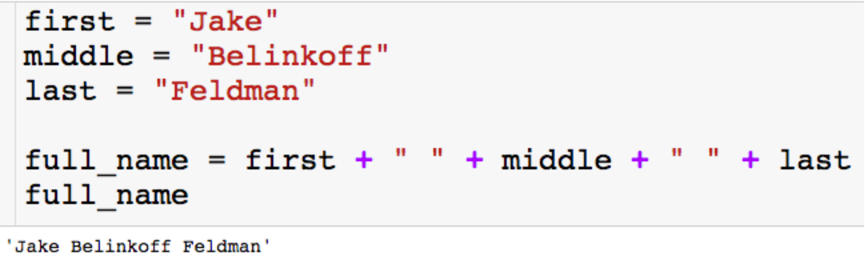
```
'JakeBelinkoffFeldman'
```

- If we want a space, we have to say so.

42

String Concatenation

- I can combine strings using the + operator.
- So the + operator between two numbers add them and the + operator between two strings concatenates them! This is called **polymorphism.**



```
first = "Jake"
middle = "Belinkoff"
last = "Feldman"

full_name = first + " " + middle + " " + last
full_name
```

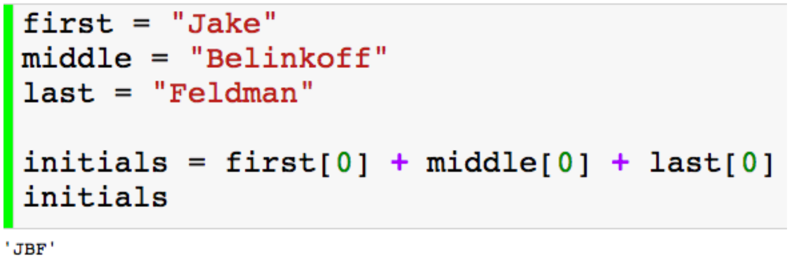
```
'Jake Belinkoff Feldman'
```

- With the space

43

String Concatenation

- I can combine strings using the + operator.
- So the + operator between two numbers add them and the + operator between two strings concatenates them! This is called **polymorphism.**



```
first = "Jake"
middle = "Belinkoff"
last = "Feldman"

initials = first[0] + middle[0] + last[0]
initials
```

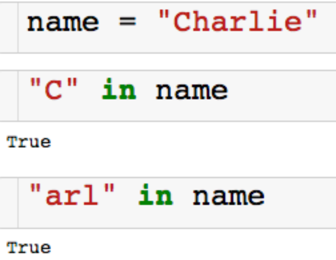
```
'JBF'
```

- Another example

44

Using In

- We can use the keyword in to check if a string is contained in another string.



```
name = "Charlie"
```

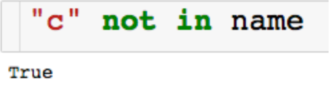
```
"C" in name
```

```
True
```

```
"arl" in name
```

```
True
```

- There is also a not in:



```
"c" not in name
```

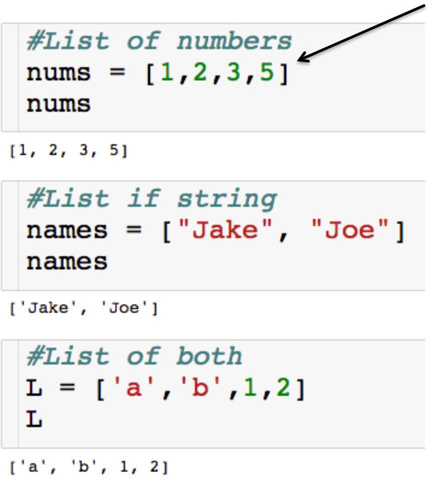
```
True
```

Lists

- Ordered collection of arbitrary objects.
  - There is a left to right ordering (just like string).
  - Can contain numbers, string, or even other lists.
- Elements accessed by offset.
  - You can fetch elements by index (just like string).
  - You can also do slicing and concatenation.
- Variable in length and arbitrarily nestable.
  - Lists can grow and shrink in-place.
  - You can have lists of lists of lists…

Lists

- Lets create our first lists



```
#List of numbers
nums = [1,2,3,5]
nums
```

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

```
#List if string
names = ["Jake", "Joe"]
names
```

```
['Jake', 'Joe']
```

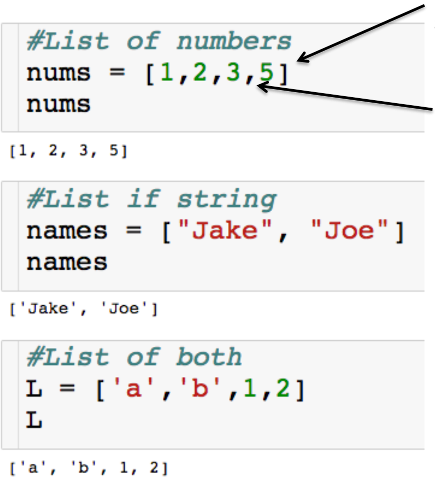
```
#List of both
L = ['a','b',1,2]
L
```

```
['a', 'b', 1, 2]
```

Elements enclosed in square brackets.

Lists

- Lets create our first lists



```
#List of numbers
nums = [1,2,3,5]
nums
```

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

```
#List if string
names = ["Jake", "Joe"]
names
```

```
['Jake', 'Joe']
```

```
#List of both
L = ['a','b',1,2]
L
```

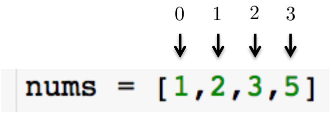
```
['a', 'b', 1, 2]
```

Elements enclosed in square brackets.

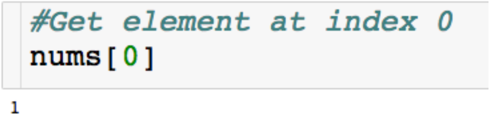
Elements separated by commas.

Indexing Lists

- Indexing for lists is very similar to strings

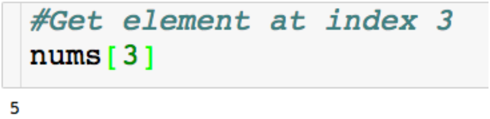


```
nums = [1,2,3,5]
```



```
#Get element at index 0
nums[0]
```

```
1
```



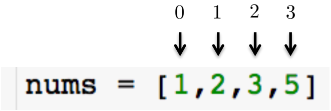
```
#Get element at index 3
nums[3]
```

```
5
```

49

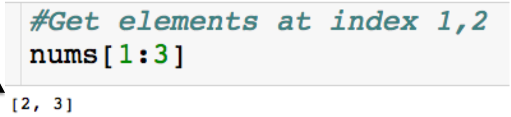
Slicing Lists

- Slicing for lists I also very similar to strings



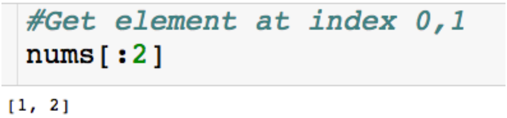
```
nums = [1,2,3,5]
```

Returns lists



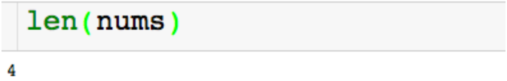
```
#Get elements at index 1,2
nums[1:3]
```

```
[2, 3]
```



```
#Get element at index 0,1
nums[:2]
```

```
[1, 2]
```



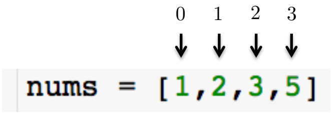
```
len(nums)
```

```
4
```

50

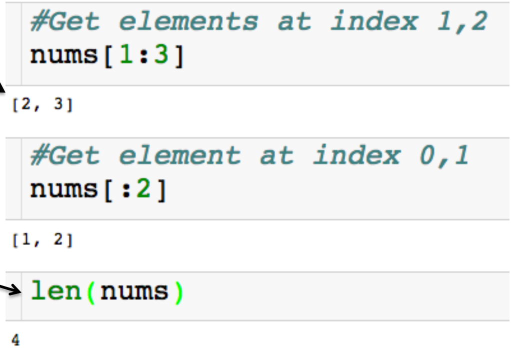
Slicing Lists

- Slicing for lists I also very similar to strings



```
nums = [1,2,3,5]
```

Returns lists



```
#Get elements at index 1,2
nums[1:3]
```

```
[2, 3]
```

```
#Get element at index 0,1
nums[:2]
```

```
[1, 2]
```

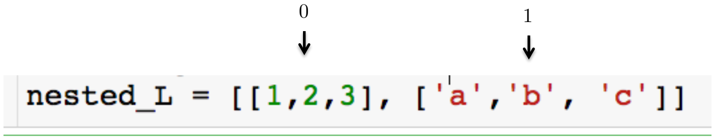
Returns # of elements in list

```
len(nums)
```

```
4
```

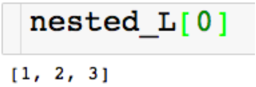
Nested Lists

- Creating a nested list:



```
nested_L = [[1,2,3], ['a','b', 'c']]
```

- There are two elements in the list nested_L.
  - There is a list of numbers in index 0.
  - There is a list of string of index 1.

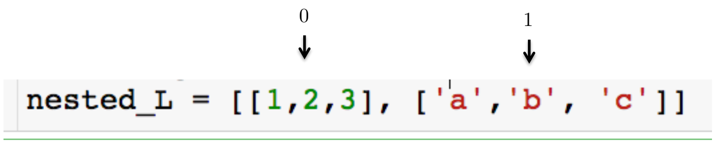


```
nested_L[0]
```

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

Indexing Nested Lists

- Creating a nested list:



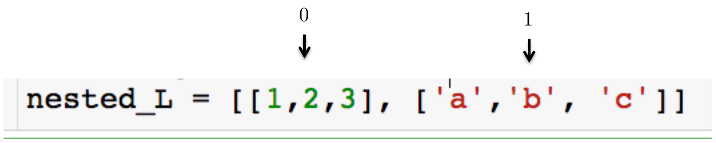
```
nested_L = [[1,2,3], ['a','b', 'c']]
```

- How do I pick out the 2 in the first list?

53

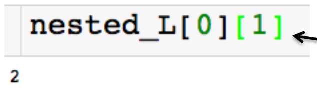
Indexing Nested Lists

- Creating a nested list:



```
nested_L = [[1,2,3], ['a','b', 'c']]
```

- How do I pick out the 2 in the first list?
  - First pick out the list of numbers, then from that pick out the



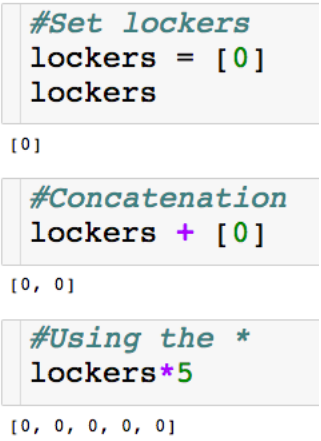
```
nested_L[0][1]
2
```

Stack the indexing

54

Polymorphism with Lists

- The + and * operator work on lists as well!



```
#Set lockers
lockers = [0]
lockers
```

```
[0]
```

```
#Concatenation
lockers + [0]
```

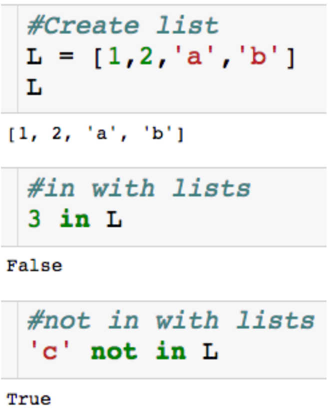
```
[0, 0]
```

```
#Using the *
lockers*5
```

```
[0, 0, 0, 0, 0]
```

Using in with Lists

- Keywords in and not in work with lists as well.



```
#Create list
L = [1,2,'a','b']
L
```

```
[1, 2, 'a', 'b']
```

```
#in with lists
3 in L
```

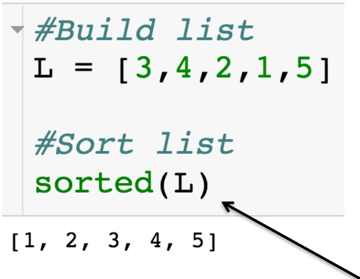
```
False
```

```
#not in with lists
'c' not in L
```

```
True
```

Sorting Lists

- We can sort lists with the built-in sorted() function.



```
#Build list
L = [3,4,2,1,5]

#Sort list
sorted(L)
```

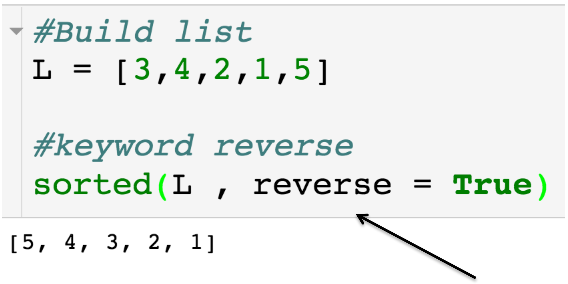
```
[1, 2, 3, 4, 5]
```

Returns sorted version of list

57

Sorting Lists

- We can sort lists with the built-in sorted() function.



```
#Build list
L = [3,4,2,1,5]

#keyword reverse
sorted(L , reverse = True)
```

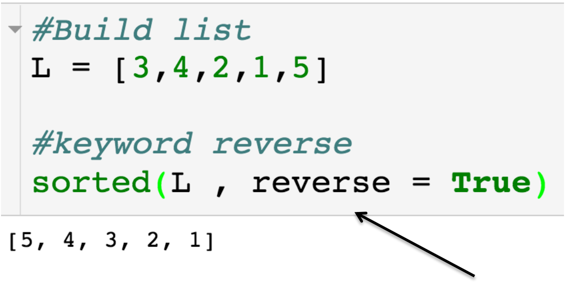
```
[5, 4, 3, 2, 1]
```

- Sort list descending
- Default is reverse = False

58

Sorting Lists

- We can sort lists with the built-in sorted() function.



```
#Build list
L = [3,4,2,1,5]

#keyword reverse
sorted(L , reverse = True)
```

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

- Sort list descending
- Default is reverse = False

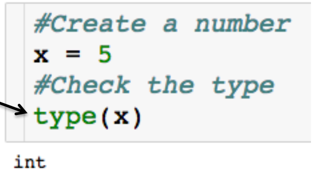
Next session we will see how to sort L “inplace”.

Intro to Python Objects – Part 2

python™

Checking the Type

- For any variable, we can check what kind of object it is:



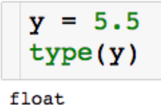
```
#Create a number
x = 5
#Check the type
type(x)
```

```
int
```

Built in type function

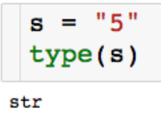
61

Why the Type Matters



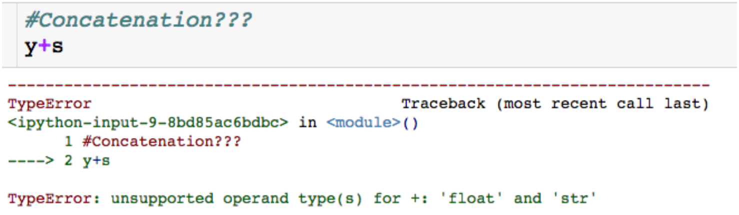
```
y = 5.5
type(y)
```

```
float
```



```
s = "5"
type(s)
```

```
str
```



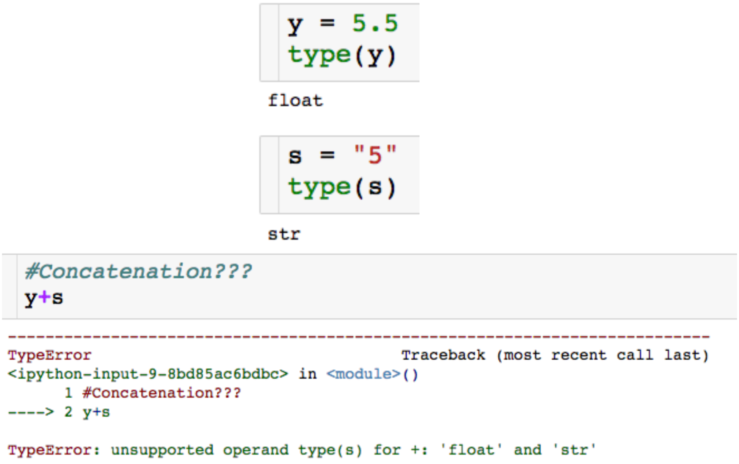
```
#Concatenation???
y+s
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-8bd85ac6bdbc> in <module>()
      1 #Concatenation???
----> 2 y+s

TypeError: unsupported operand type(s) for +: 'float' and 'str'
```

We can’t concatenate a string and a number…and we shouldn’t be able

62

Why the Type Matters



```
y = 5.5
type(y)
```

```
float
```

```
s = "5"
type(s)
```

```
str
```

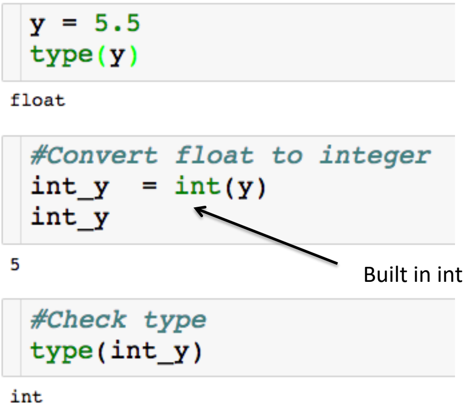
```
#Concatenation???
y+s
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-8bd85ac6bdbc> in <module>()
      1 #Concatenation???
----> 2 y+s

TypeError: unsupported operand type(s) for +: 'float' and 'str'
```

- type() can be helpful for debugging
- Another reason to have descriptive variable names

63

Converting Types



```
y = 5.5
type(y)
```

```
float
```

```
#Convert float to integer
int_y  = int(y)
int_y
```

```
5
```

Built in int() function

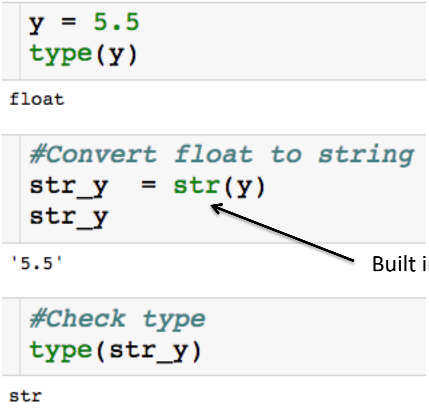
```
#Check type
type(int_y)
```

```
int
```

- int() is one way to perform a floor operation

64

Converting Types



```
y = 5.5
type(y)
```

```
float
```

```
#Convert float to string
str_y  = str(y)
str_y
```

```
'5.5'
```

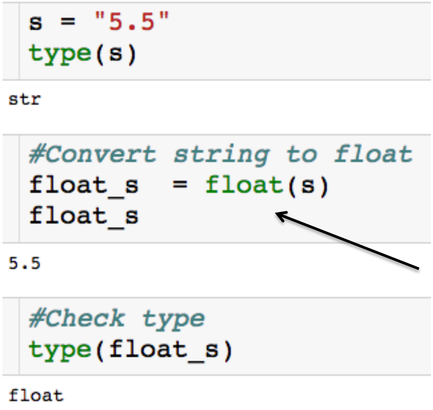
Built in str() function

```
#Check type
type(str_y)
```

```
str
```

65

Converting Types



```
s = "5.5"
type(s)
```

```
str
```

```
#Convert string to float
float_s  = float(s)
float_s
```

```
5.5
```

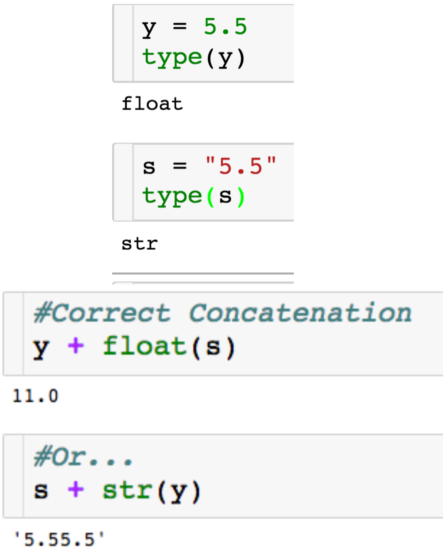
Built in float() function

```
#Check type
type(float_s)
```

```
float
```

66

Why the Type Matters



```
y = 5.5
type(y)
```

```
float
```

```
s = "5.5"
type(s)
```

```
str
```

```
#Correct Concatenation
y + float(s)
```

```
11.0
```

```
#Or...
s + str(y)
```

```
'5.55.5'
```

67

Digging Deeper into Python Objects

- Every Python Object is either mutable or immutable
  - **Mutable:** Can be changed once created - a list L can have its first element replaced.
  - **Immutable:** Can't be changed once creating – a string S cannot have its first letter changed.

68

Digging Deeper into Python Objects

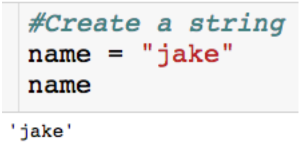
- Every Python Object is either mutable or immutable
  - **Mutable:** Can be changed once created - a list L can have its first element replaced.
  - **Immutable:** Can't be changed once creating – a string S cannot have its first letter changed.

What we know so far:

- **Numbers = Immutable**
- **String = Immutable**
- **Lists = Mutable**

69

Example of Immutability



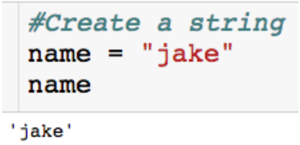
```
#Create a string
name = "jake"
name
```

```
'jake'
```

Let's say I want to change the first letter of name to a "J"

70

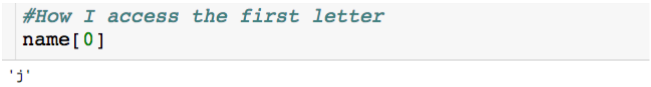
Example of Immutability



```
#Create a string
name = "jake"
name
```

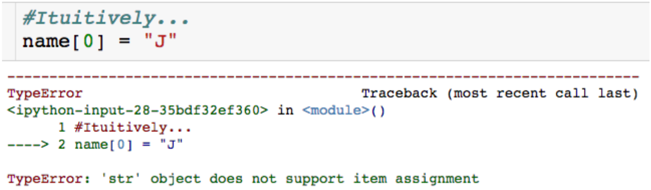
```
'jake'
```

Let’s say I want to change the first letter of name to a “J”



```
#How I access the first letter
name[0]
```

```
'j'
```

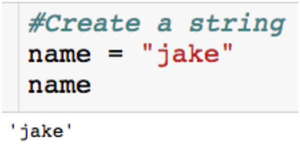


```
#Ituitively...
name[0] = "J"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-35bdf32ef360> in <module>()
      1 #Ituitively...
----> 2 name[0] = "J"

TypeError: 'str' object does not support item assignment
```

Can’t change name once it is created!

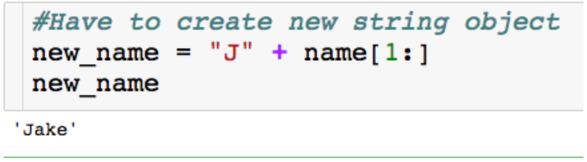
Example of Immutability



```
#Create a string
name = "jake"
name
```

```
'jake'
```

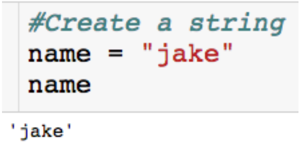
Let’s say I want to change the first letter of name to a “J”



```
#Have to create new string object
new_name = "J" + name[1:]
new_name
```

```
'Jake'
```

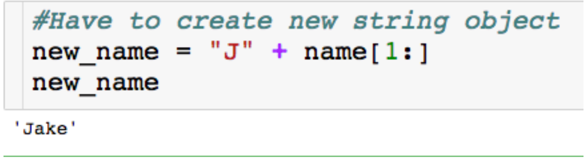
Example of Immutability



```
#Create a string
name = "jake"
name
```

```
'jake'
```

Let’s say I want to change the first letter of name to a “J”



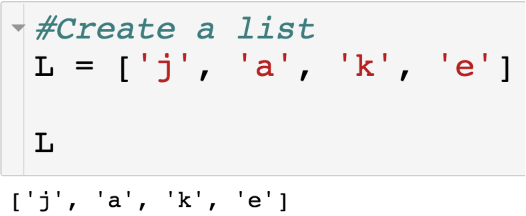
```
#Have to create new string object
new_name = "J" + name[1:]
new_name
```

```
'Jake'
```

We will see an easier way to do this…

73

Example of Mutability



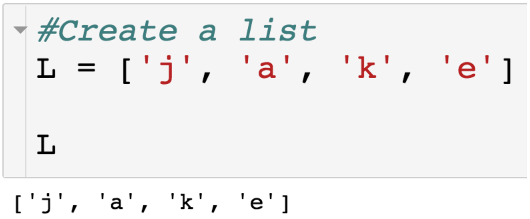
```
#Create a list
L = ['j', 'a', 'k', 'e']

L
```

```
['j', 'a', 'k', 'e']
```

Let’s say I want to change the string in index 0 to a “J”.

74

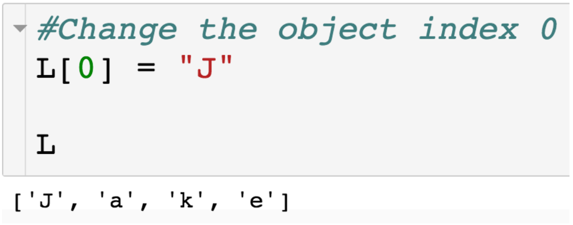
Example of Mutability



```
#Create a list
L = ['j', 'a', 'k', 'e']

L
```

```
['j', 'a', 'k', 'e']
```

Let’s say I want to change the string in index 0 to a “J”.



```
#Change the object index 0
L[0] = "J"

L
```

```
['J', 'a', 'k', 'e']
```

Since lists are mutable, we change an any part of list after it has been created.

75