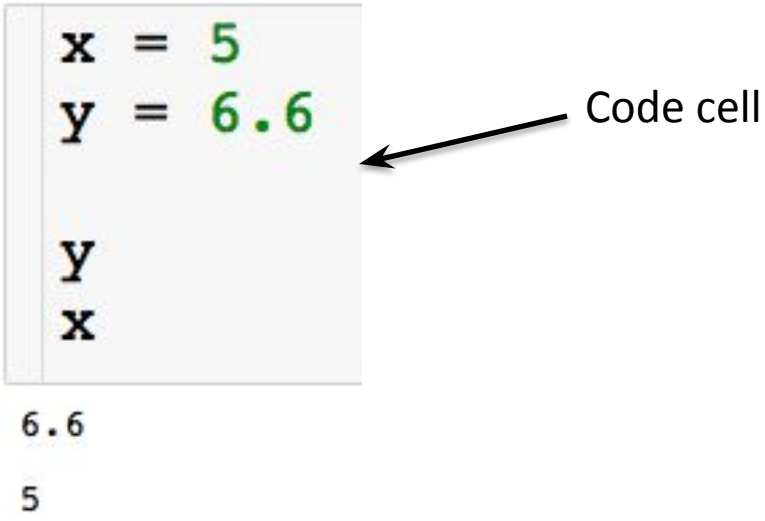
Intro to Python Objects – Part 1

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.

Creating Variables

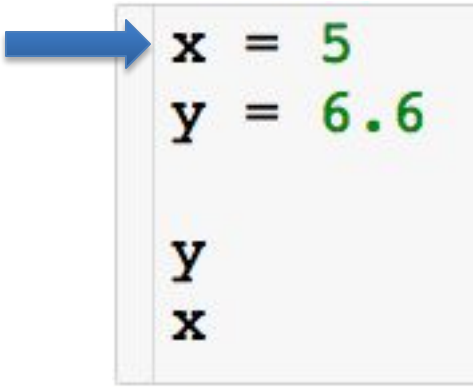
Let’s create our first variables



The code executes from top to bottom

Creating Variables

Let's create our first variables



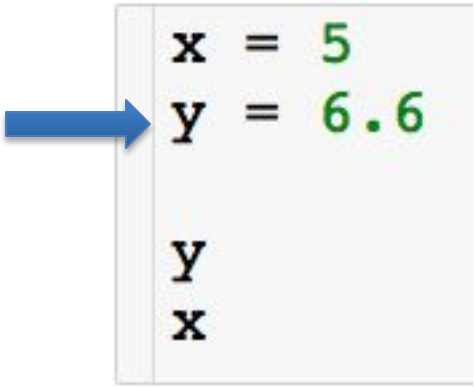
```
x = 5
y = 6.6

y
x
```

```
6.6
5
```

$$x = 5$$

Creating Variables

Let's create our first variables. These variables with be numbers.



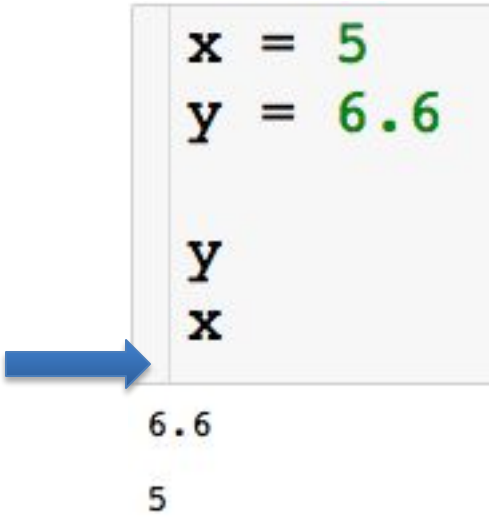
```
x = 5
y = 6.6

y
x
```

```
6.6
5
```

$$x = 5$$
$$y = 6.6$$

Creating Variables

Let's create our first variables



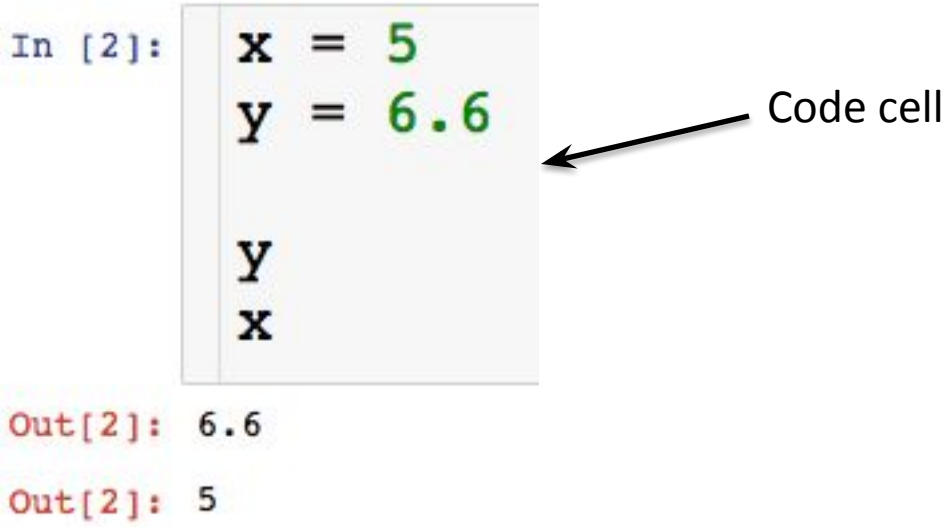
```
x = 5
y = 6.6

y
x
```

```
6.6
5
```

$$x = 5$$
$$y = 6.6$$

Creating Variables

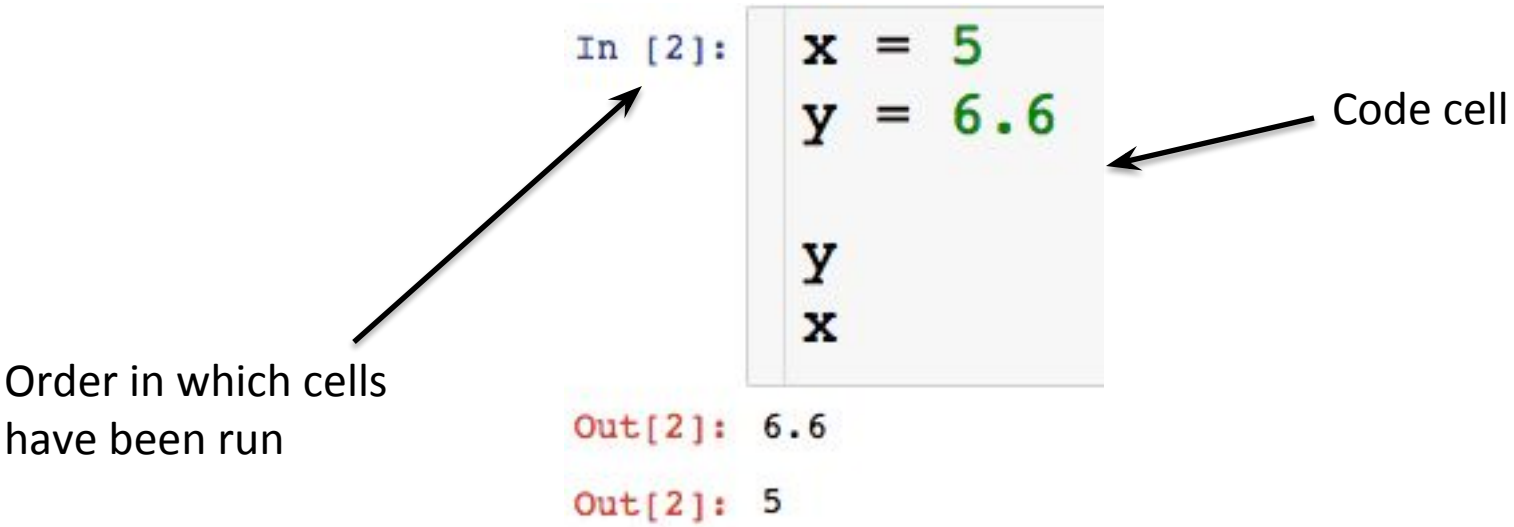
Let's create our first variables



A couple of things about Jupyter

Creating Variables

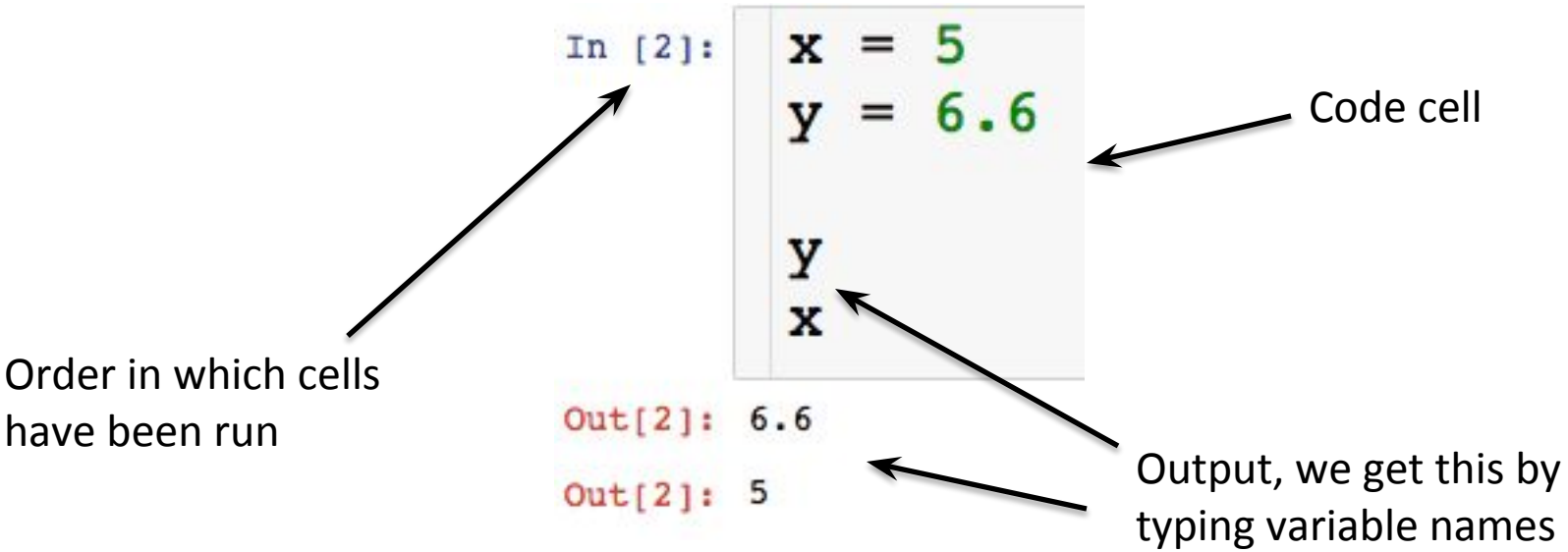
Let's create our first variables



A couple of things about Jupyter

Creating Variables

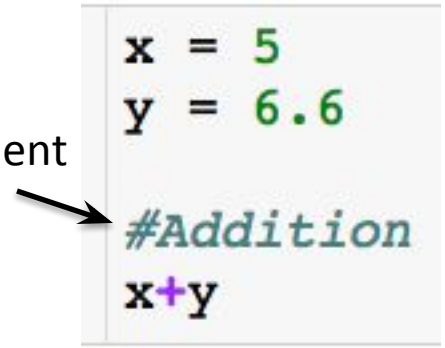
Let's create our first variables



A couple of things about Jupyter

Arithmetic Operations

= comment



```
x = 5
y = 6.6

#Addition
x+y
```

```
11.6
```

Arithmetic Operations

```
x = 5
y = 6.6

#Addition
x+y
```

```
11.6
```

```
x = 5
y = 6.6

#Substraction
x-y
```

```
11.6
```

Arithmetic Operations

```
x = 5
y = 6.6

#Addition
x+y
```

```
11.6
```

```
x = 5
y = 6.6

#Substraction
x-y
```

```
11.6
```

```
x = 5
y = 6.6

#Multiplication
x*y
```

```
33.0
```

Arithmetic Operations

```
x = 5
y = 6.6

#Addition
x+y
```

```
11.6
```

```
x = 5
y = 6.6

#Substraction
x-y
```

```
11.6
```

```
x = 5
y = 6.6

#Multiplication
x*y
```

```
33.0
```

```
x = 5

#Exponentiating
x**2
```

```
25
```

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

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.

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.

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
```

$\text{num\_quarters} = 7$

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
```

$\text{num\_quarters} = 7$

$\text{num\_nickels} = 10$

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
```

$$\text{num\_quarters} = 7$$
$$\text{num\_nickels} = 10$$
$$\text{num\_nickels} = 5$$
$$\text{total\_change} = 2.75$$

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.

$$\text{num\_quarters} = 7$$
$$\text{num\_nickels} = 10$$
$$\text{num\_nickels} = 5$$
$$\text{total\_change} = 2.75$$

Using Variables

You will often find yourself updating variables:

```
count = 0
```

Some other code executes…want to add 1 to count

Using Variables

You will often find yourself updating variables:

```
count = 0
```

Some other code executes…want to add 1 to count

```
count = count + 1
```

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
```

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.
- Let us check if specified conditions are true

Booleans

- Creating boolean variable:

```
boolean_var = True

boolean_var
```

```
True
```

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

```
boolean_var*5
```

```
5
```

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'
```

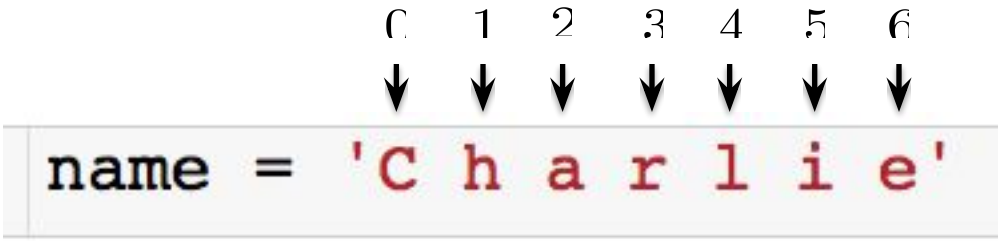
```
name = "Charlie"
name
```

```
'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**



(pretend there aren't spaces between the letters)

Slicing single characters through e index:

```
name[0]
```

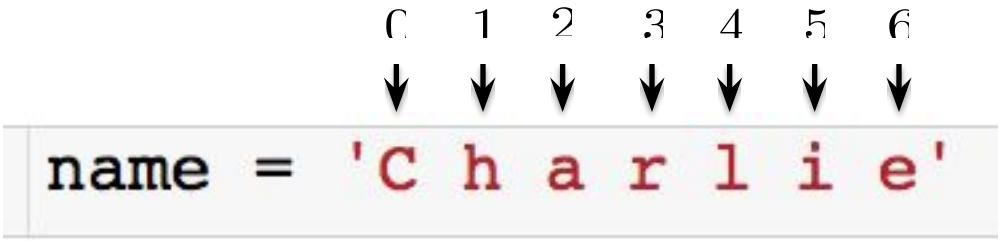
```
'C'
```

```
name[6]
```

```
'e'
```

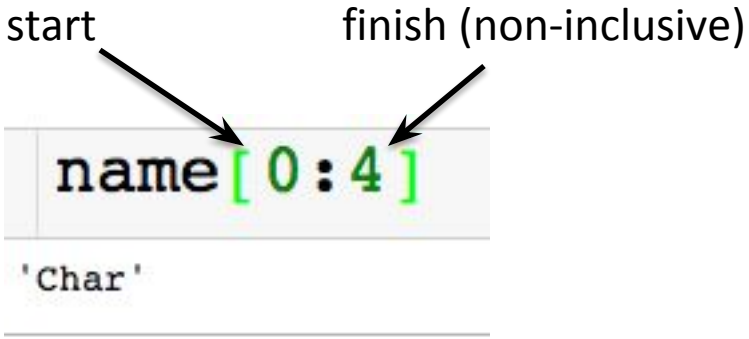
Slicing Strings

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



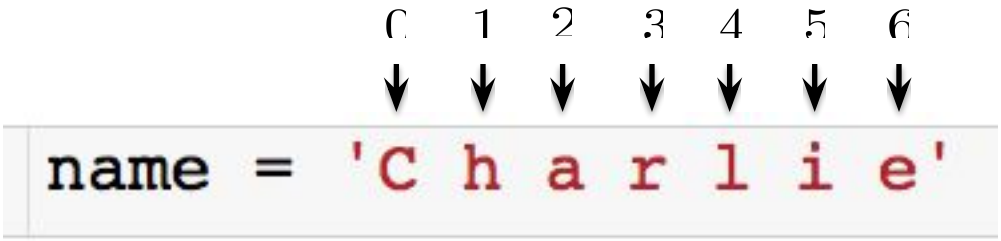
(pretend there aren't spaces between the letters)

Slicing contiguous characters:



Slicing Strings

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



(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!

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

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.**

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.

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

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

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:

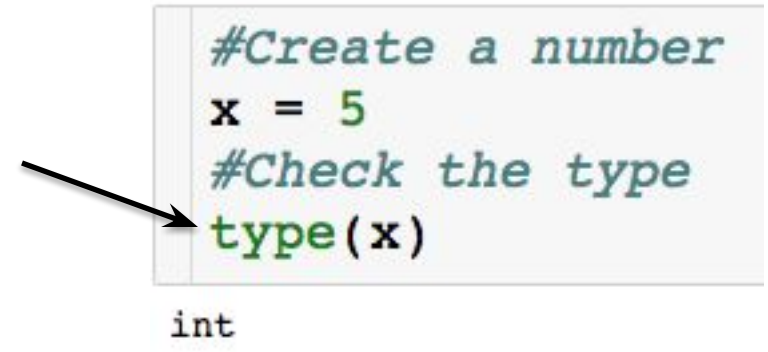
```
"c" not in name
```

```
True
```

Checking the Type

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

Built in type function



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

```
int
```

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

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

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

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
```

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
```

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'
```

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.

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**

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"

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…

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".

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.

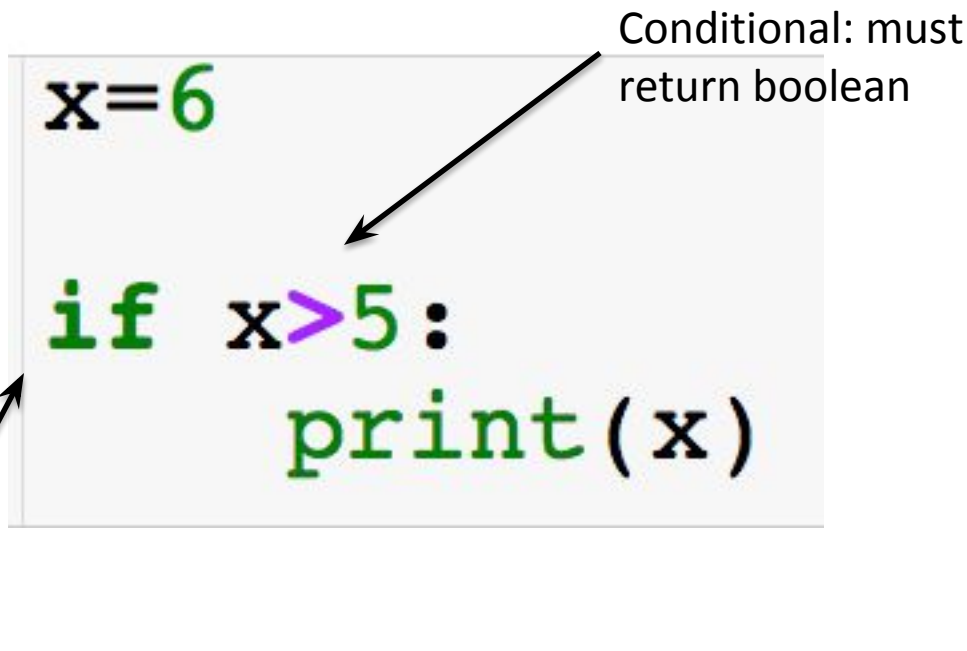
Conditionals and Loops

If Statement

Execute a certain block of code only if a certain condition is true

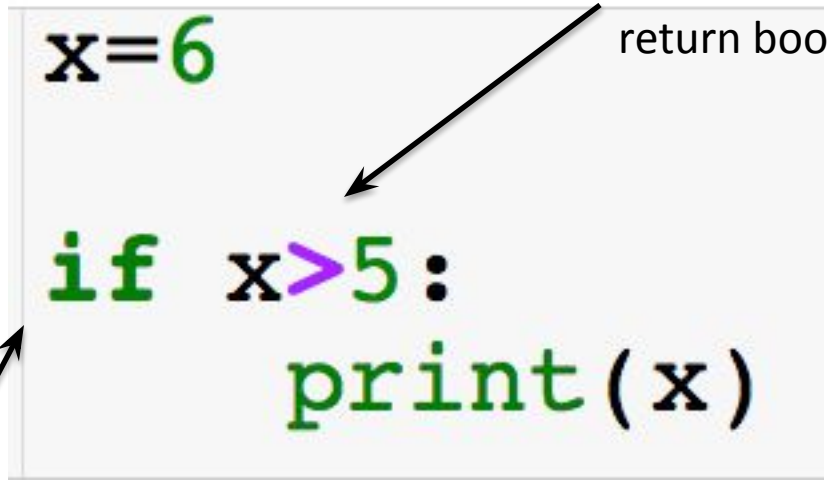
If Statement

Execute a certain block of code only if a certain condition is true



If Statement

Execute a certain block of code only if a certain condition is true



Conditional: must return boolean

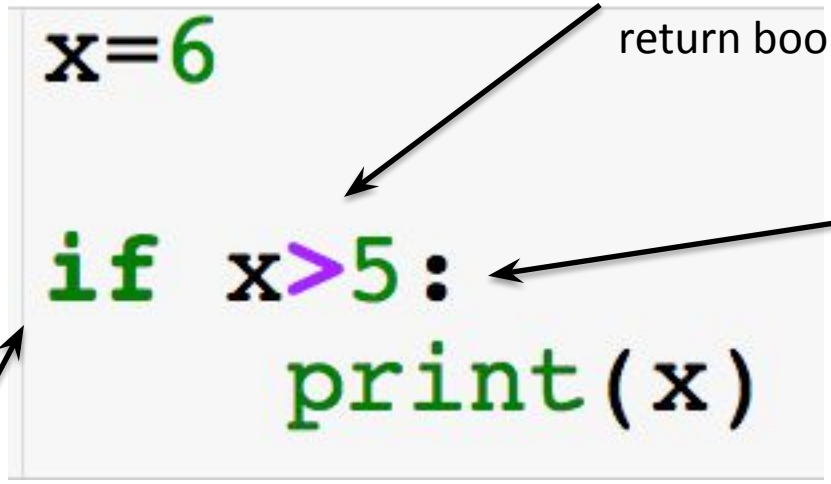
```
x = 6
#is x greater than 5?
x>5
```

```
True
```

If statements always begin with an “if”

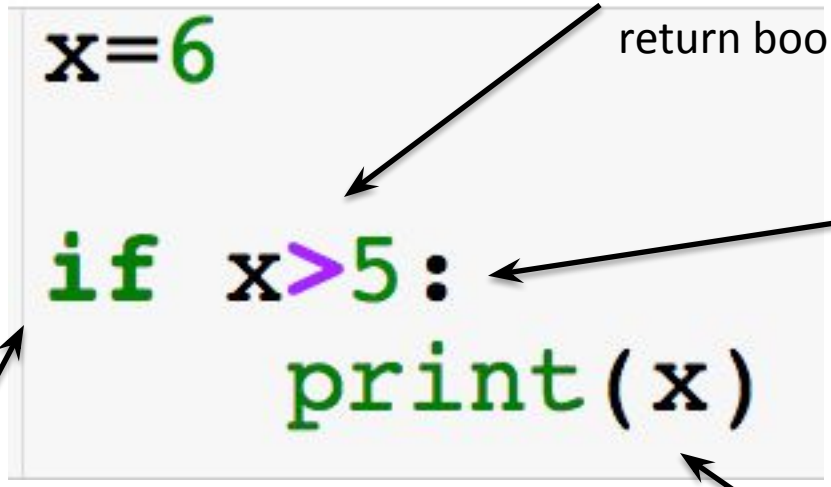
If Statement

Execute a certain block of code only if a certain condition is true



If Statement

Execute a certain block of code only if a certain condition is true



Conditional: must return boolean

":" ends statement

If statements always begin with an "if"

If condition is true, then indented line of code will run

If Statement

```
x = 6
#is x greater than 5?
if x>5:
    print(x)
```

```
6
```

If + Else Statement

Else statement allows us to specify separate directions if the condition with the if is False.

If + Else Statement

Else statement allows us to specify separate directions if the condition with the if is False.

```
x = 4
#is x greater than 5?
if x>5:
    print(x)
else:
    print("Below 5")
```

- Else **always** executes when the if condition is False.
- Match if + else by indentation
- Else is optional, meaning an if statement doesn't have to have an accompanying else.

If + Else Statement

```
x = 4
#is x greater than 5?
if x>5:
    print(x)
else:
    print("Below 5")
```

```
Below 5
```

If condition is False, so code inside of else executes

If + Elif + Else Statement

How do I specify multiple conditions?...

If + Elif + Else Statement

How do I specify multiple conditions?...elif

If + Elif + Else Statement

How do I specify multiple conditions?...elif

```
x = 'killer rabbit'
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
else:
    print("Run!!!!")
```

- Comes between if and else
- Can have arbitrary number of elifs
- First if/elif condition that is true is executed and only this block is executed
  - If no if/elif condition is true then else is executed
- Indentation matches up if/elif/else statements

If + Elif + Else Statement

```
x = 'killer rabbit'
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
else:
    print("Run!!!!")
```

```
x == 'roger'
```

```
False
```

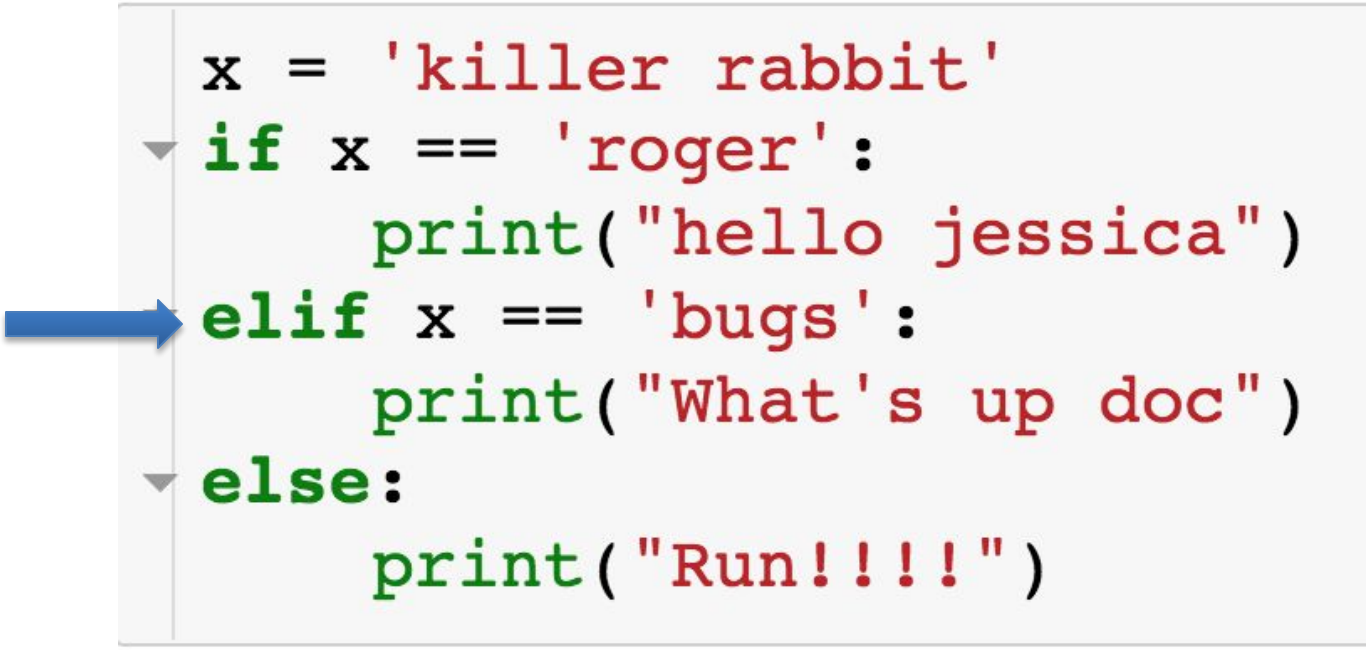
If + Elif + Else Statement

```
x = 'killer rabbit'
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
else:
    print("Run!!!!")
```

```
x == 'bugs'
```

```
False
```

If + Elif + Else Statement



```
x = 'killer rabbit'
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
else:
    print("Run!!!!")
```

```
x == 'bugs'
```

```
False
```

Since all if/elif conditions were False, else statement executes

If + Elif + Else Statement

```
x = 'killer rabbit'
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
else:
    print("Run!!!!")
```

```
Run!!!!
```

If + Elif + Else Statement

We can just have an if + elif, with no else:

```
x = 'killer rabbit'
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
```

```
```

If + Elif + Else Statement

We can just have an if + elif, with no else:

```
x = 'killer rabbit'
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
```

Nothing is printed since neither if not elif condition is true

If + Elif + Else Statement

Example with multiple elif conditions:

```
x = 'killer rabbit'
season = "winter"
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
elif season == "winter":
    print("no bunnies")
else:
    print("Run!!!!")
```

If + Elif + Else Statement

Example with multiple elif conditions:

```
x = 'killer rabbit'
season = "winter"
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
elif season == "winter":
    print("no bunnies")
else:
    print("Run!!!!")
```

If + Elif + Else Statement

Example with multiple elif conditions:

```
x = 'killer rabbit'
season = "winter"
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
elif season == "winter":
    print("no bunnies")
else:
    print("Run!!!!")
```

If + Elif + Else Statement

Example with multiple elif conditions:

```
x = 'killer rabbit'
season = "winter"
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
elif season == "winter":
    print("no bunnies")
else:
    print("Run!!!!")
```

If + Elif + Else Statement

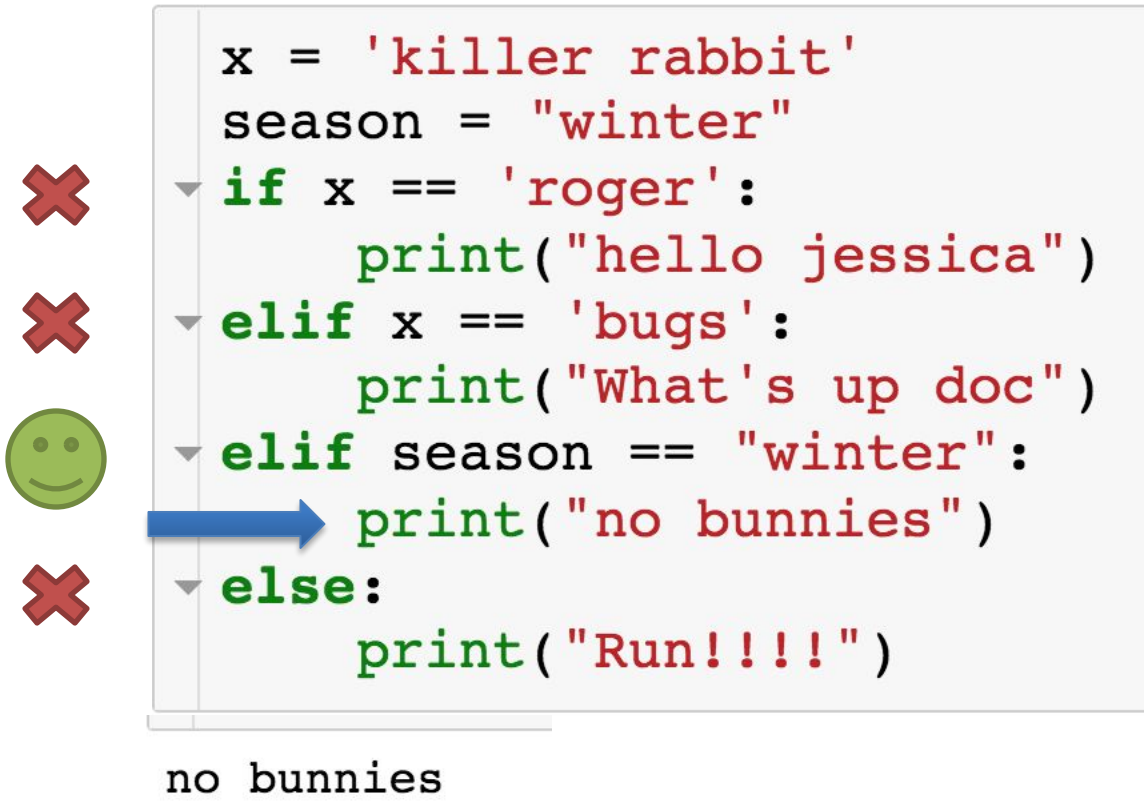
Example with multiple elif conditions:

```
x = 'killer rabbit'
season = "winter"
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
elif season == "winter":
    print("no bunnies")
else:
    print("Run!!!!")
```

```
no bunnies
```

If + Elif + Else Statement

Example with multiple elif conditions:



```
x = 'killer rabbit'
season = "winter"
if x == 'roger':
    print("hello jessica")
elif x == 'bugs':
    print("What's up doc")
elif season == "winter":
    print("no bunnies")
else:
    print("Run!!!!")
```

```
no bunnies
```

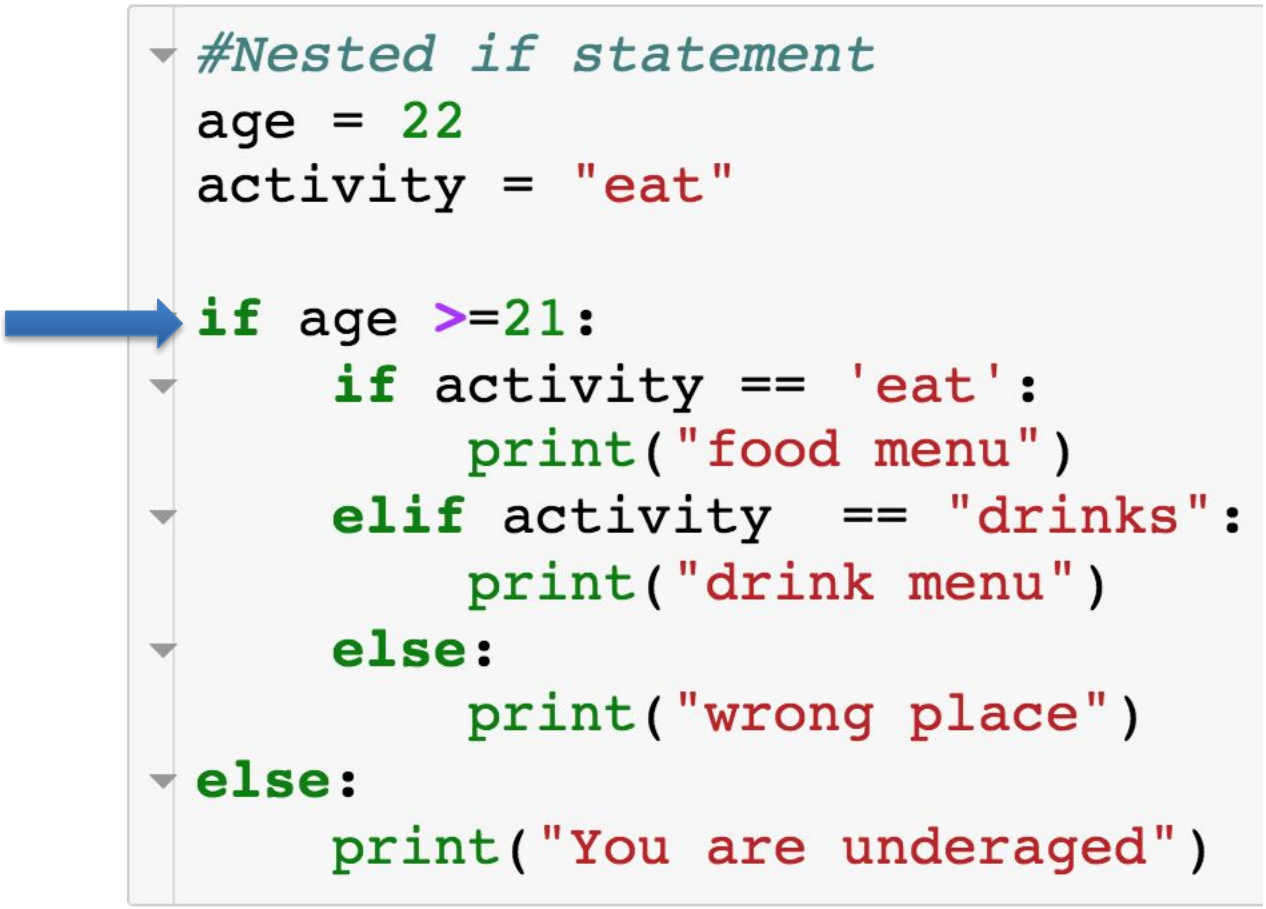
Nested Conditionals

We can have if statements nested within if statements:

```
#Nested if statement
age = 22
activity = "eat"

if age >=21:
    if activity == 'eat':
        print("food menu")
    elif activity  == "drinks":
        print("drink menu")
    else:
        print("wrong place")
else:
    print("You are underaged")
```

Nested Conditionals

We can have if statements nested within if statements:



```
#Nested if statement
age = 22
activity = "eat"

if age >=21:
    if activity == 'eat':
        print("food menu")
    elif activity  == "drinks":
        print("drink menu")
    else:
        print("wrong place")
else:
    print("You are underaged")
```

Nested Conditionals

We can have if statements nested within if statements:

```
#Nested if statement
age = 22
activity = "eat"

if age >=21:
    if activity == 'eat':
        print("food menu")
    elif activity  == "drinks":
        print("drink menu")
    else:
        print("wrong place")
else:
    print("You are underaged")
```

Nested Conditionals

We can have if statements nested within if statements:

```
#Nested if statement
age = 22
activity = "eat"

if age >=21:
    if activity == 'eat':
        print("food menu")
    elif activity  == "drinks":
            print("drink menu")
    else:
            print("wrong place")
else:
    print("You are underaged")
```

Nested Conditionals

We can have if statements nested within if statements:

```
#Nested if statement
age = 22
activity = "eat"

if age >=21:
    if activity == 'eat':
        print("food menu")
    elif activity  == "drinks":
            print("drink menu")
    else:
            print("wrong place")
else:
    print("You are underaged")
```

```
food menu
```

Nested Conditionals

```
#Nested Conditionals
bat_avg = 0.312
hr = 39
rbi = 103
if bat_avg> 0.33:
    print("all-star")
else:
    if hr>40 or rbi>150:
        print("all-star")
    elif bat_avg>0.3 and hr>30 and rbi>100:
        print("all-star")
    else:
        print("nope")
```

Nested Conditionals

```
#Nested Conditionals
bat_avg = 0.312
hr = 39
rbi = 103
if bat_avg> 0.33:
    print("all-star")
else:
    if hr>40 or rbi>150:
        print("all-star")
    elif bat_avg>0.3 and hr>30 and rbi>100:
        print("all-star")
    else:
        print("nope")
```

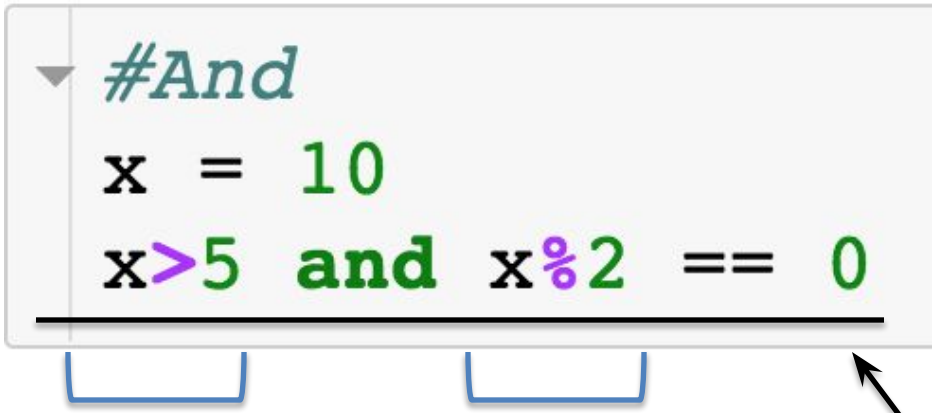
```
all-star
```

Specifying Multiple Conditions

How do you specify multiple conditions in a conditional statements?

Specifying Multiple Conditions - and

How do you specify multiple conditions in a conditional statements?



Specifying Multiple Conditions - and

How do you specify multiple conditions in a conditional statements?

```
#And
x = 10
x>5 and x%2 == 0
```

Condition 1 Condition 2

AND

- All conditions must evaluate to True
- Can have arbitrary number of conditions

Specifying Multiple Conditions - and

How do you specify multiple conditions in a conditional statements?

```
#And
x = 10
x>5 and x%2 == 0
```

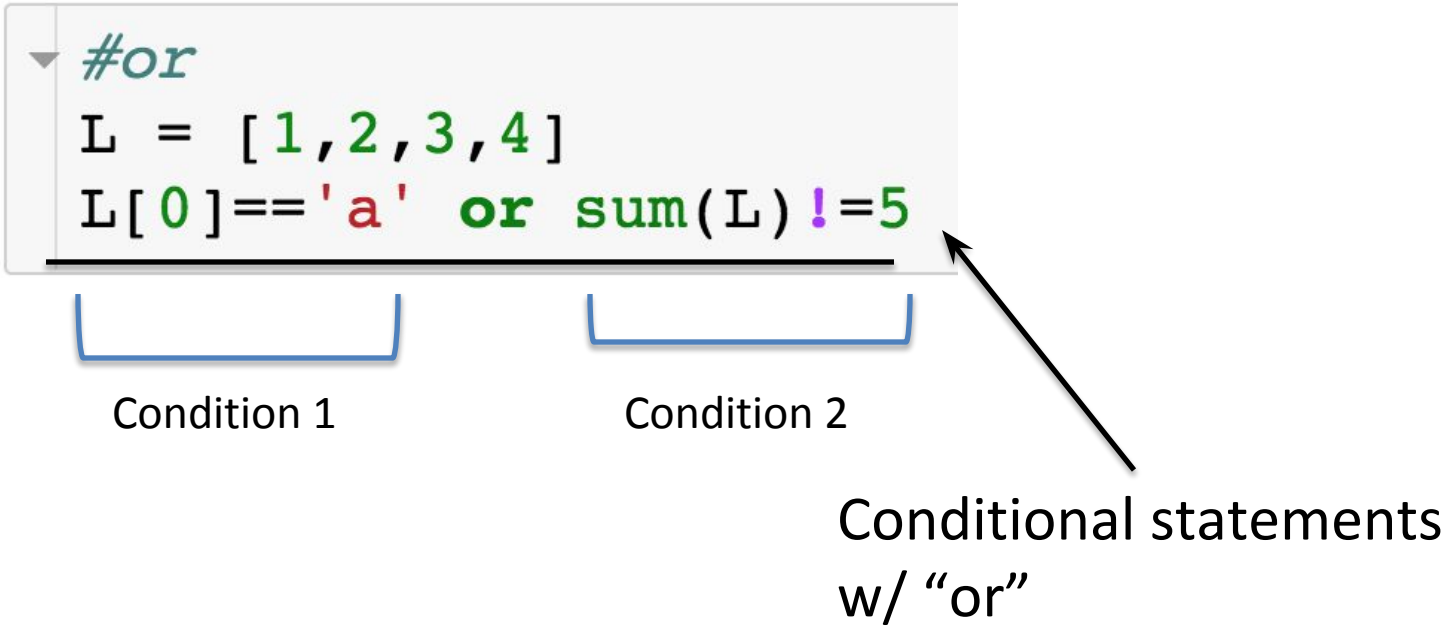
```
True
```

```
#Another example
name = 'Jake'
name[0]=='J' and len(name)>5
```

```
False
```

Specifying Multiple Conditions - or

How do you specify multiple conditions in a conditional statements?



Specifying Multiple Conditions - or

How do you specify multiple conditions in a conditional statements?

```
#or
L = [1,2,3,4]
L[0]=='a' or sum(L)!=5
```

Condition 1 Condition 2

OR

- At least one condition must evaluate to True
- Can have arbitrary number of conditions

Specifying Multiple Conditions

How do you specify multiple conditions in a conditional statements?

```
#or
L = [1,2,3,4]
L[0]=='a' or sum(L)!=5
```

```
True
```

```
#Another example
name = 'Jake'
name[0]=='J' or y in name
```

```
True
```

```
#And + or
(name=="Joe" or 'a' in name) and 5 not in L
```

```
True
```

For Loops

Use cases of for loops:

- Iterate over the elements of a list or string
    - For each element, perform some sort of operation - count or sum
- Perform some action a specified number of times
    - Deal out 5 cards
    - Have a student go through all 100 lockers

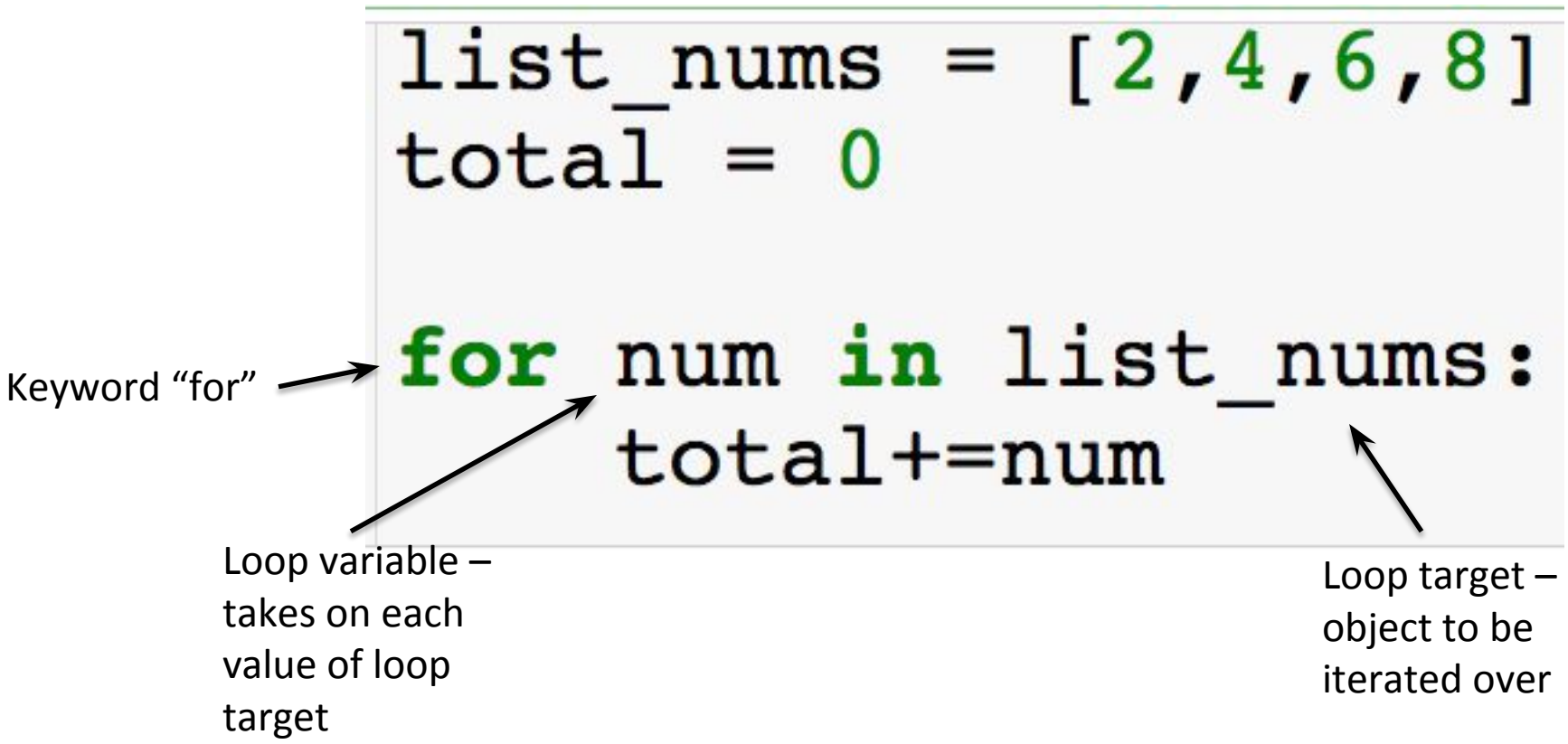
For Loops w/ List

- Compute sum of elements of a list
- Use for loop to iterate over the elements of the list

```
list_nums = [2,4,6,8]
total = 0

for num in list_nums:
    total+=num
```

For Loops w/ List

- Compute sum of elements of a list
- Use for loop to iterate over the elements of the list



For Loops w/ List

Can call loop variable whatever you want

$$num = 2$$
$$total = 0$$

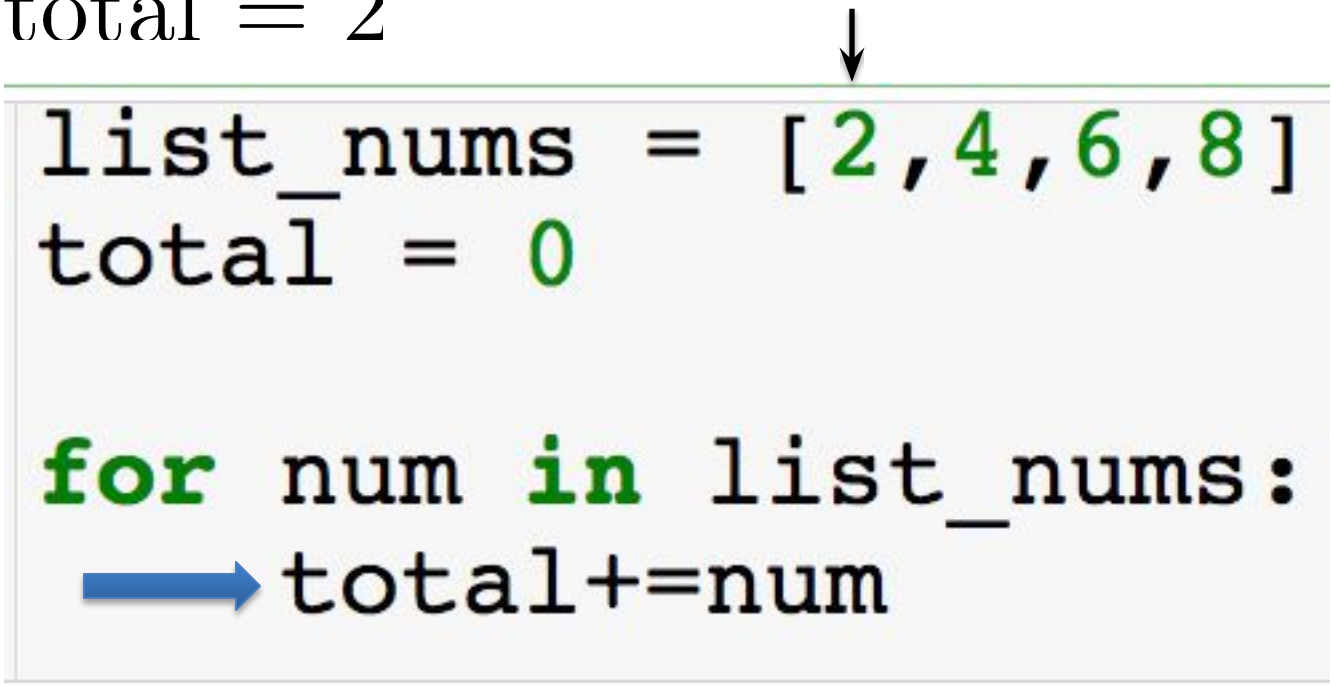
```
list_nums = [2,4,6,8]
total = 0

for num in list_nums:
    total+=num
```

Block of code to run each iteration of for loop

For Loops w/ List

$\text{num} = 2$

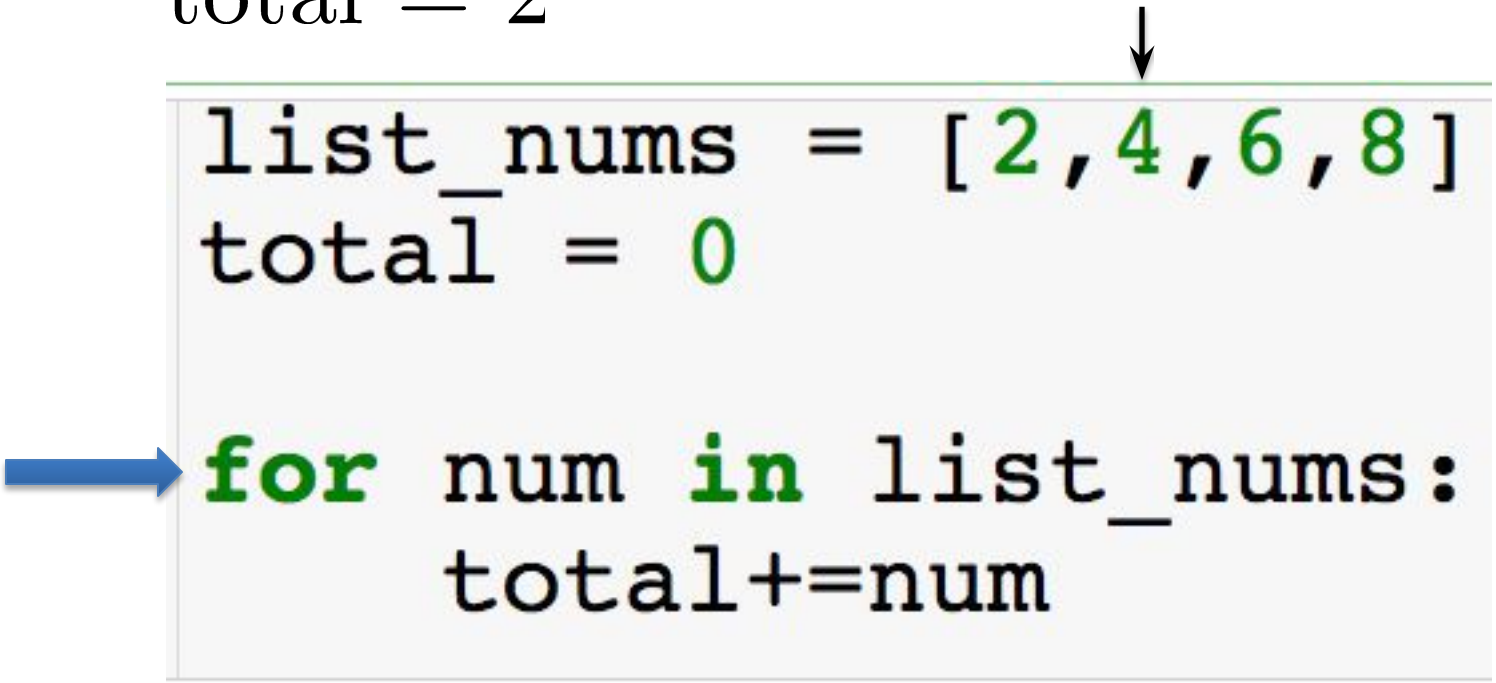
$\text{total} = 2$



For Loops w/ List

$\text{num} = 4$

$\text{total} = 2$

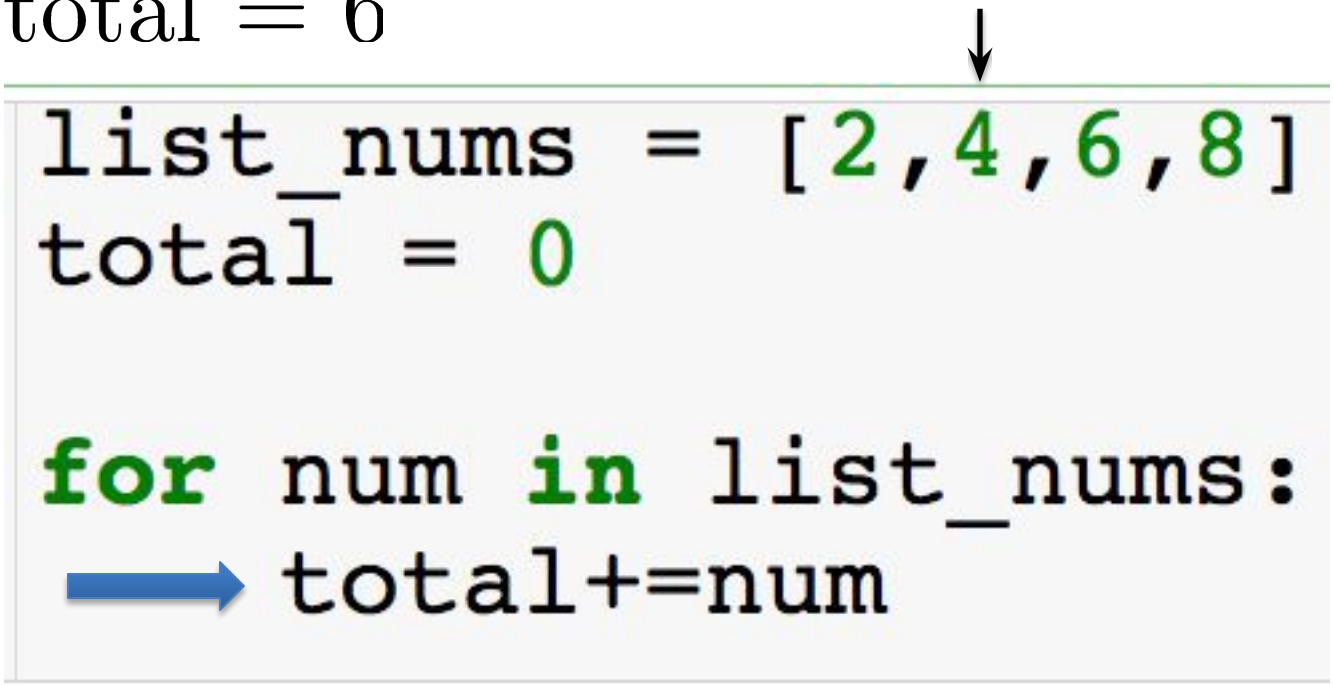


```
list_nums = [2,4,6,8]
total = 0

for num in list_nums:
    total+=num
```

For Loops w/ List

$\text{num} = 4$

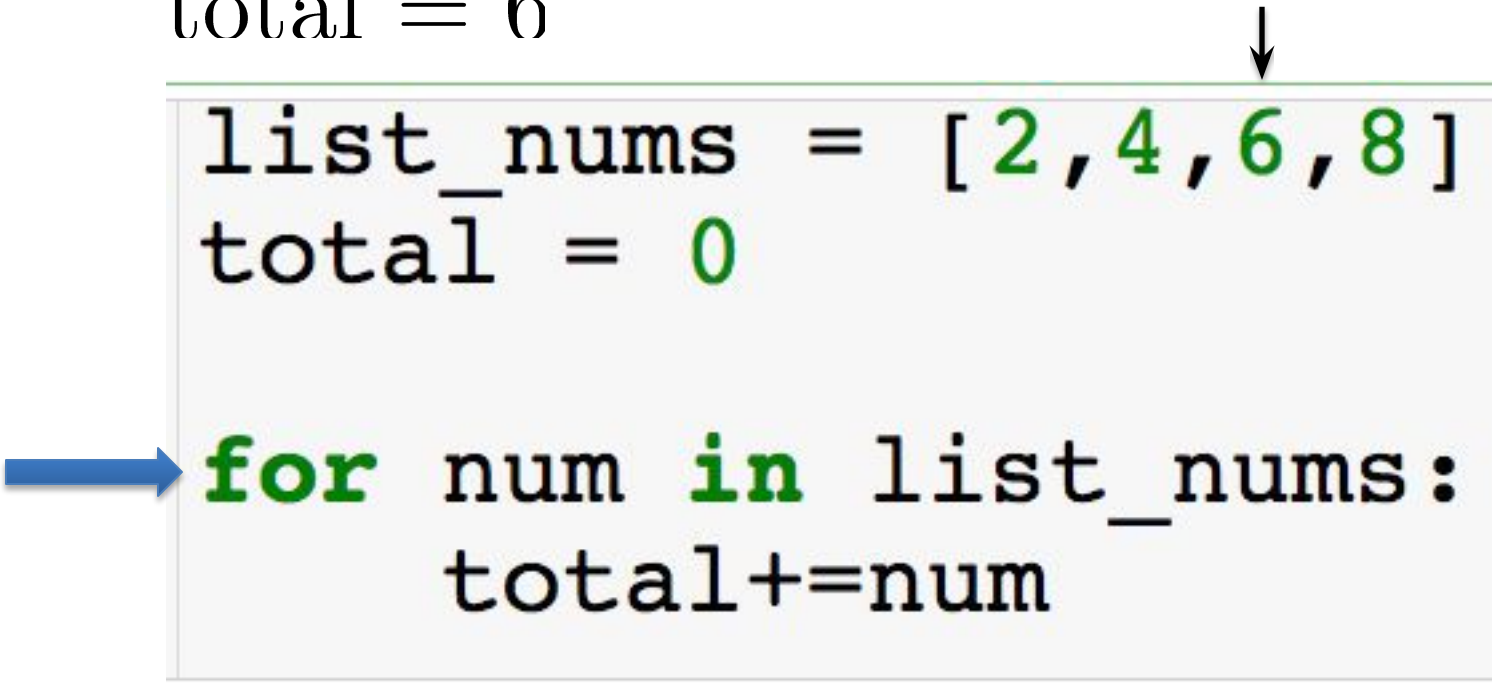
$\text{total} = 6$



For Loops w/ List

$\text{num} = 6$

$\text{total} = 6$

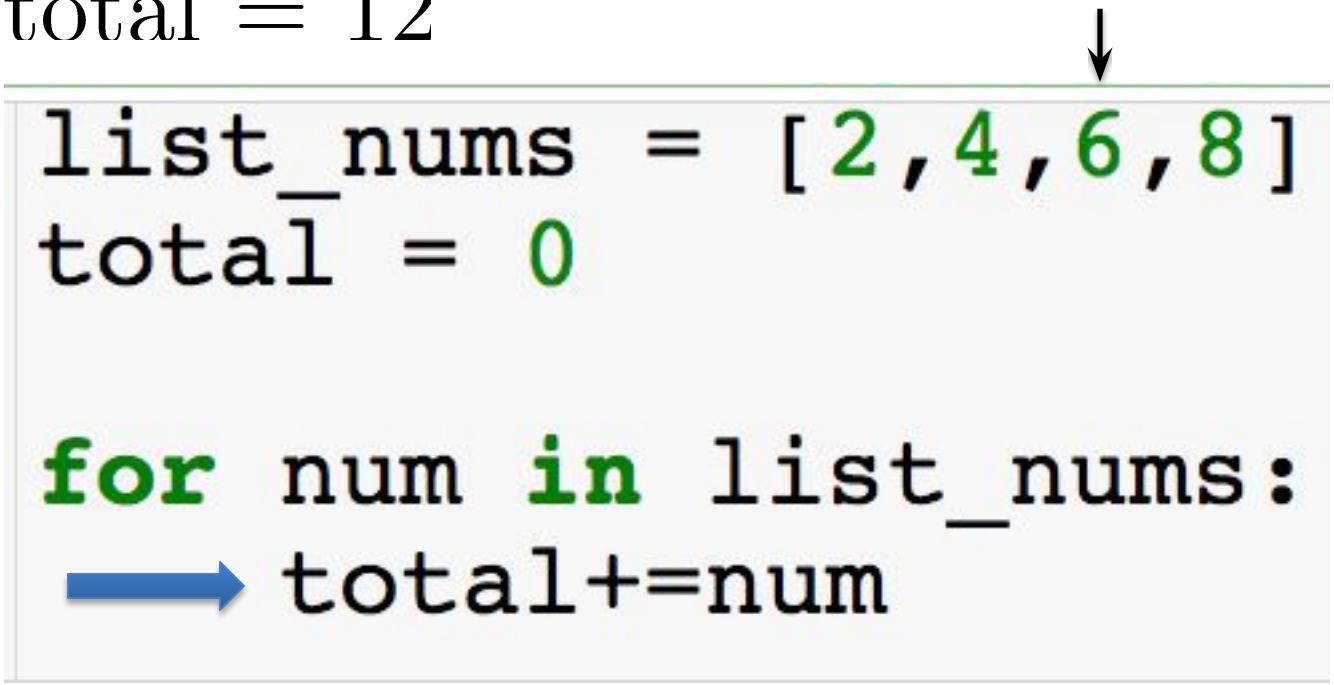


```
list_nums = [2,4,6,8]
total = 0

for num in list_nums:
    total+=num
```

For Loops w/ List

$\text{num} = 6$

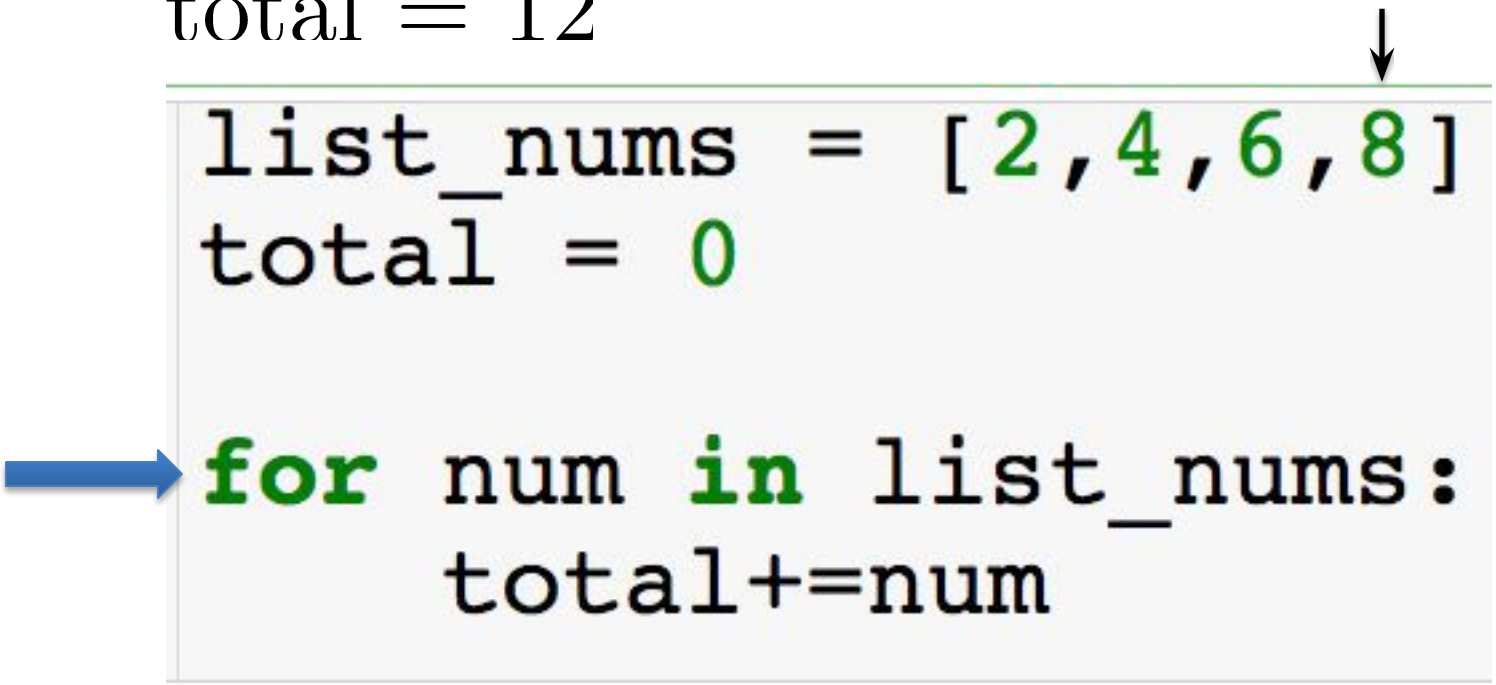
$\text{total} = 12$



For Loops w/ List

$\text{num} = 8$

$\text{total} = 12$

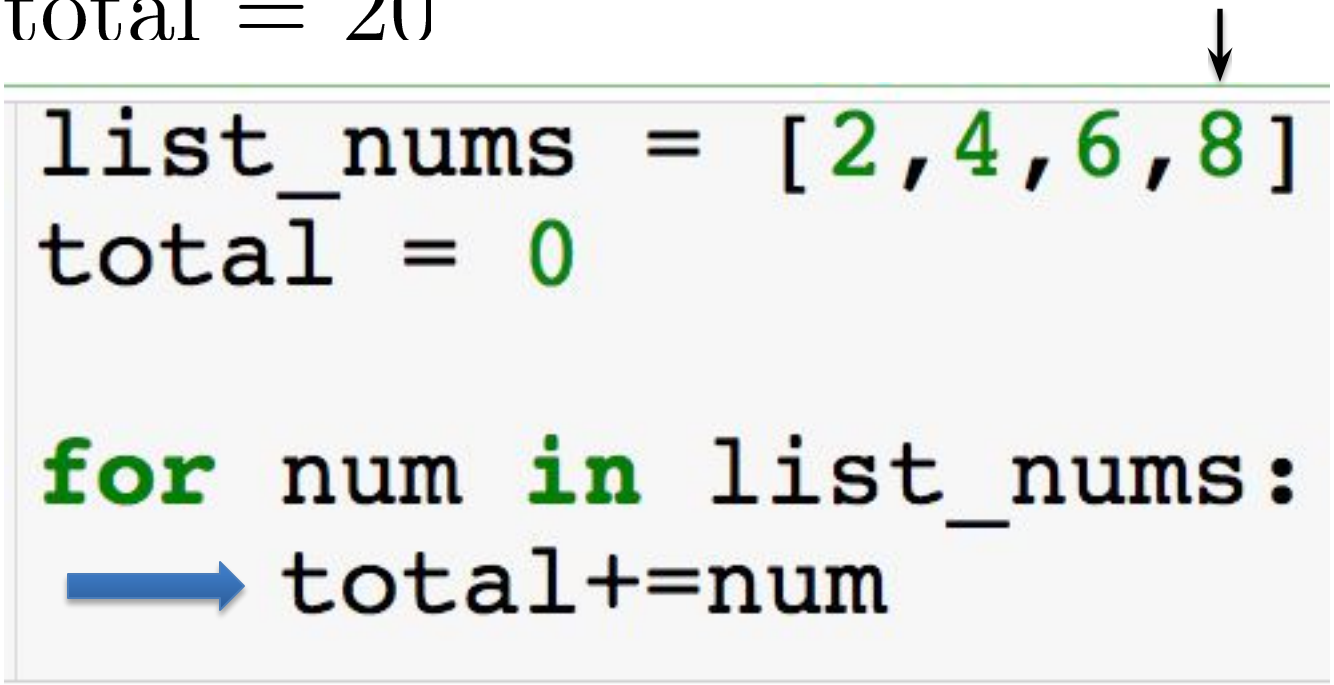


```
list_nums = [2,4,6,8]
total = 0

for num in list_nums:
    total+=num
```

For Loops w/ List

$$\text{num} = 8$$

$$\text{total} = 20$$



For Loops w/ Strings

- Now the loop target is a string
- Compute the number of words in sentence

```
sentence = "Charlie likes walks"
count = 0
for c in sentence:
    if c== " ":
        count+=1

total_words = count+1
```

For Loops w/ Strings

$c = \text{“C”}$

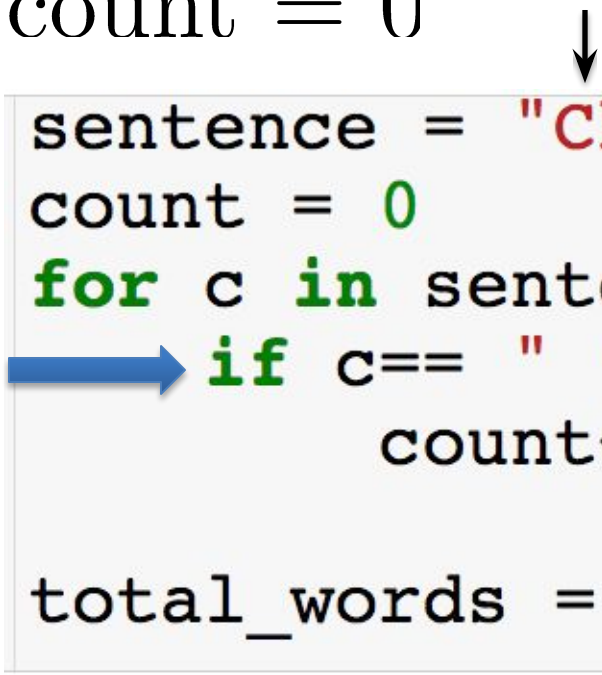
$\text{count} = 0$

```
sentence = "Charlie likes walks"
count = 0
for c in sentence:
    if c== " ":
        count+=1

total_words = count+1
```

For Loops w/ Strings

$c = \text{“C”}$

$\text{count} = 0$



```
sentence = "Charlie likes walks"
count = 0
for c in sentence:
    if c== " ":
        count+=1

total_words = count+1
```

For Loops w/ Strings

$c = \text{“h”}$

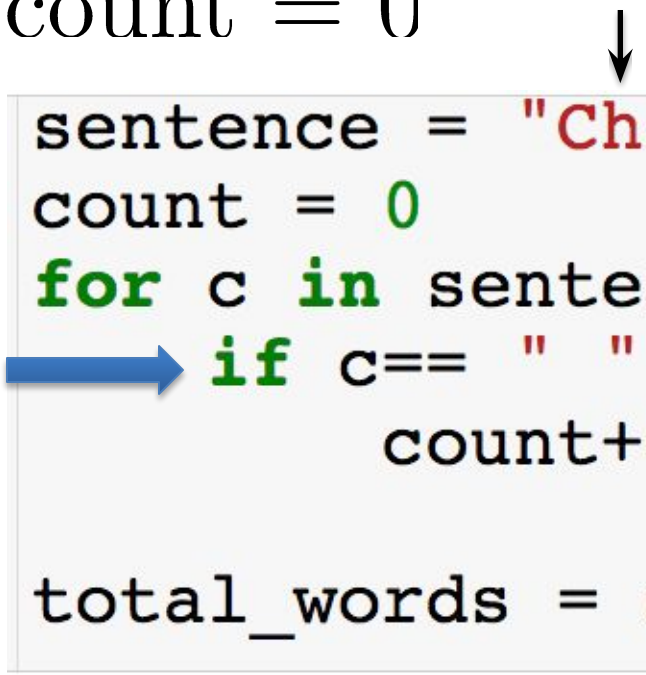
$\text{count} = 0$

```
sentence = "Charlie likes walks"
count = 0
for c in sentence:
    if c== " ":
        count+=1

total_words = count+1
```

For Loops w/ Strings

$c = \text{“h”}$

$\text{count} = 0$



```
sentence = "Charlie likes walks"
count = 0
for c in sentence:
    if c== " ":
        count+=1

total_words = count+1
```

For Loops w/ Strings

$c = \text{“e”}$

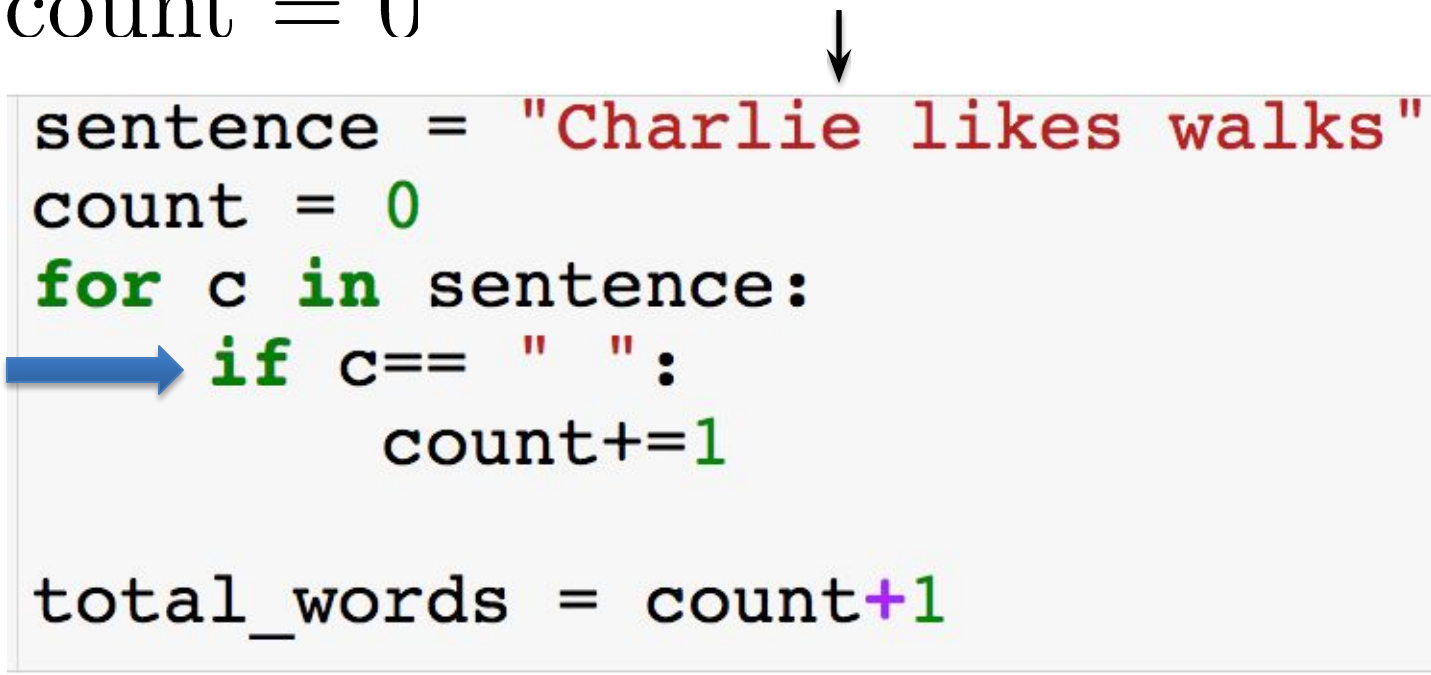
$\text{count} = 0$

```
sentence = "Charlie likes walks"
count = 0
for c in sentence:
    if c== " ":
        count+=1

total_words = count+1
```

For Loops w/ Strings

$c = \text{“e”}$

$\text{count} = 0$



```
sentence = "Charlie likes walks"
count = 0
for c in sentence:
    if c== " ":
        count+=1

total_words = count+1
```

For Loops w/ Strings

$c = \text{“ ”}$

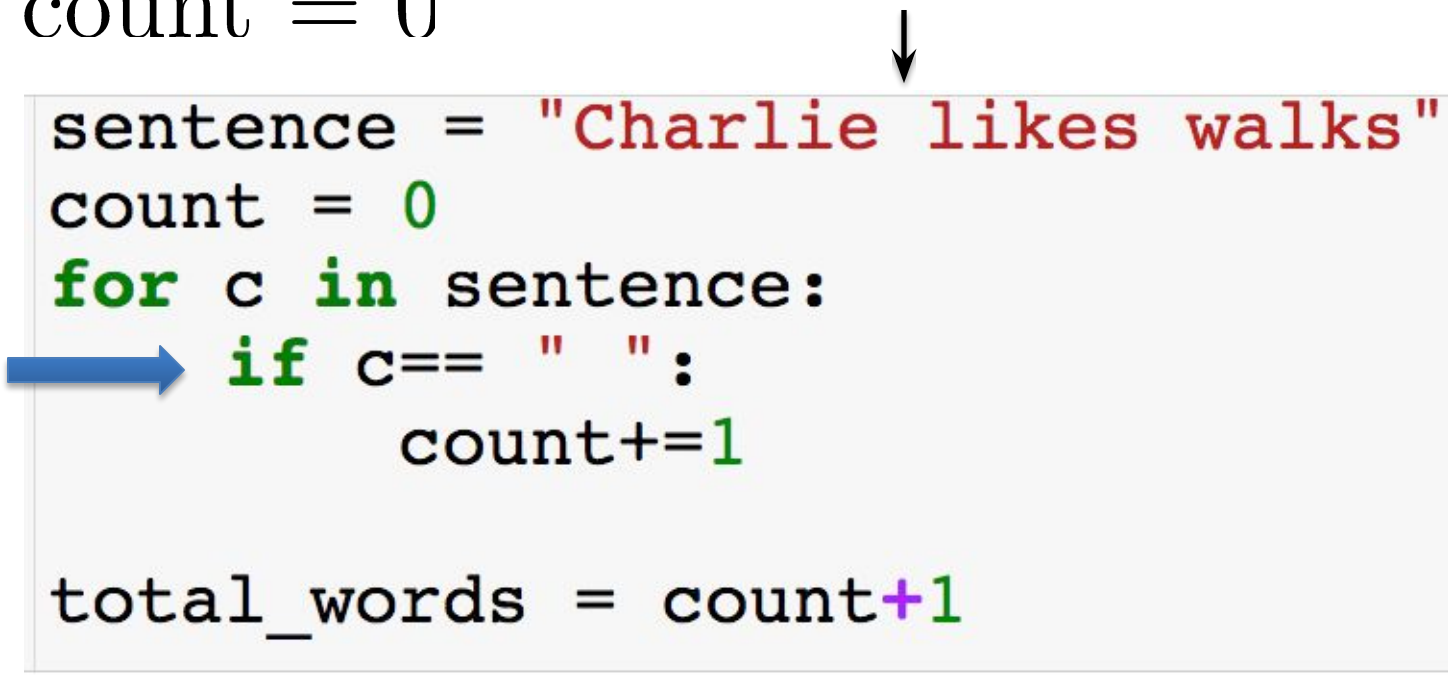
$\text{count} = 0$

```
sentence = "Charlie likes walks"
count = 0
for c in sentence:
    if c== " ":
        count+=1

total_words = count+1
```

For Loops w/ Strings

$c = \text{“ ”}$

$\text{count} = 0$

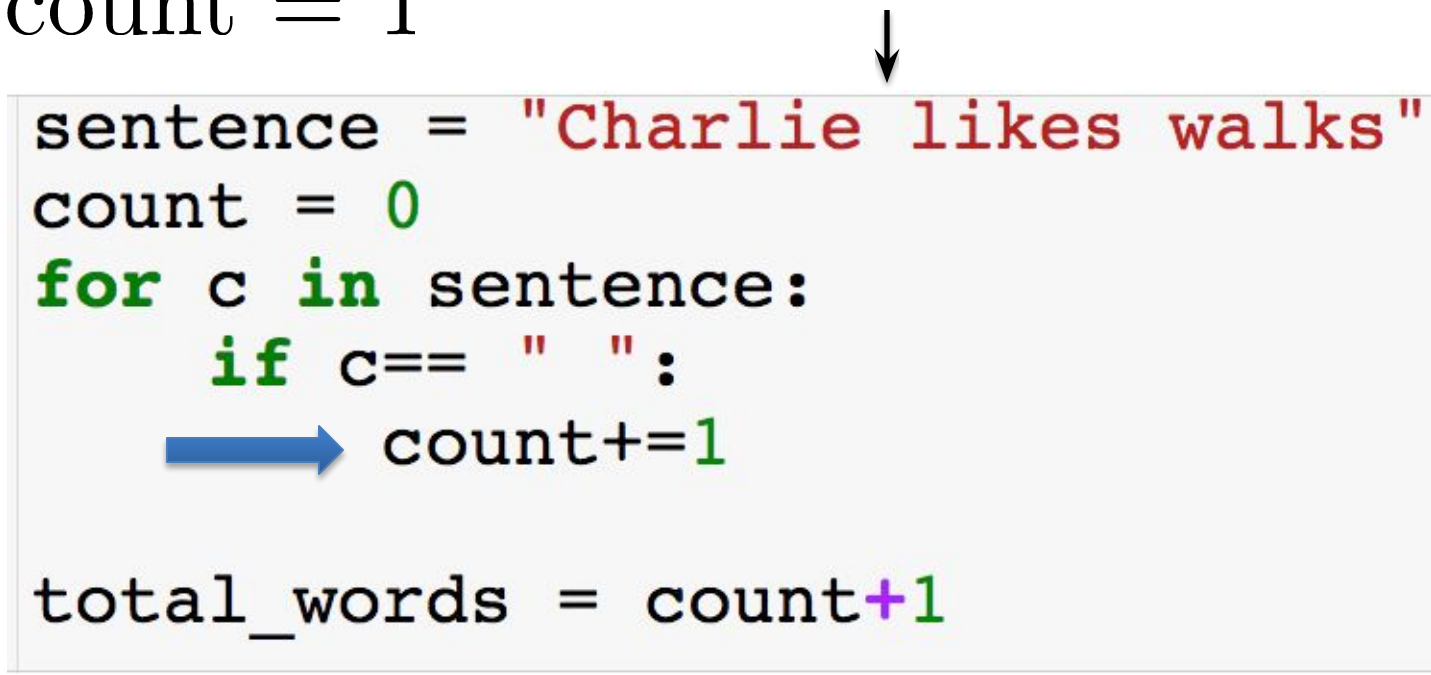


```
sentence = "Charlie likes walks"
count = 0
for c in sentence:
    if c== " ":
        count+=1

total_words = count+1
```

For Loops w/ Strings

$c = \text{“ ”}$

$\text{count} = 1$



For Loops w/ Strings

$c =$ “l”

$\text{count} = 1$

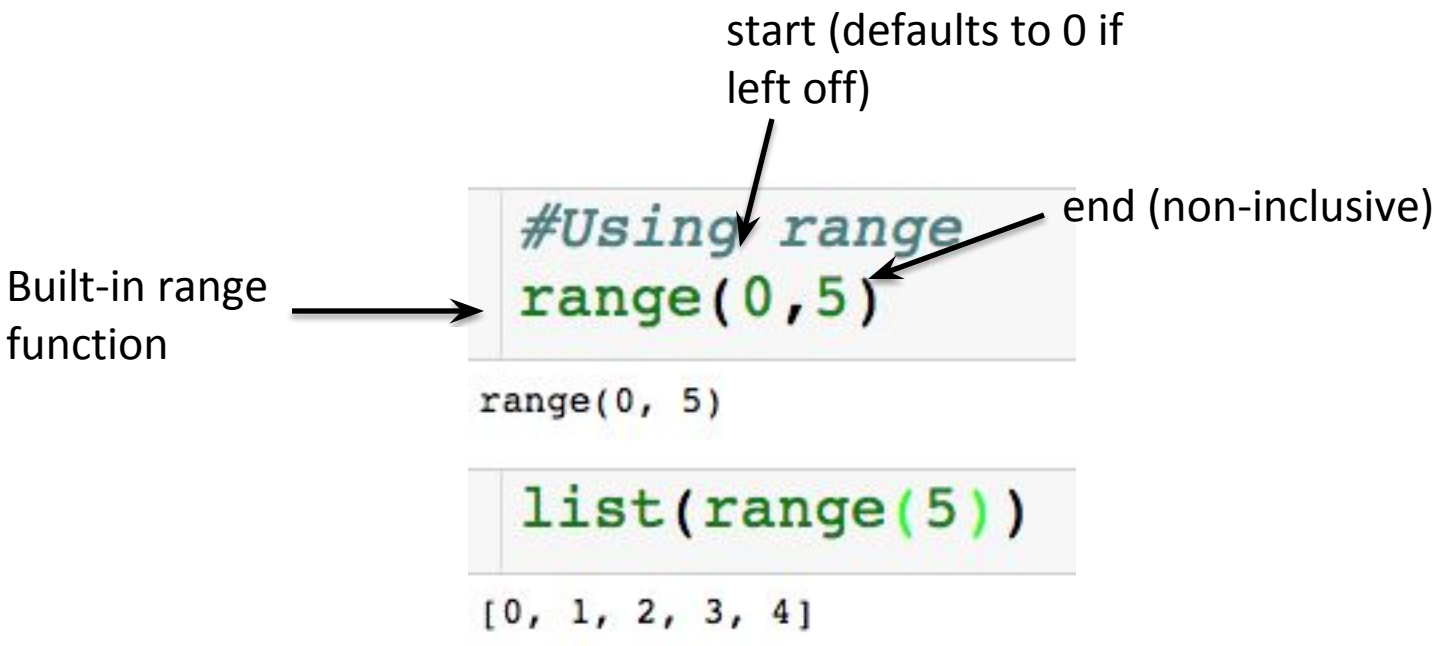
```
sentence = "Charlie likes walks"
count = 0
for c in sentence:
    if c== " ":
        count+=1

total_words = count+1
```

And so on…

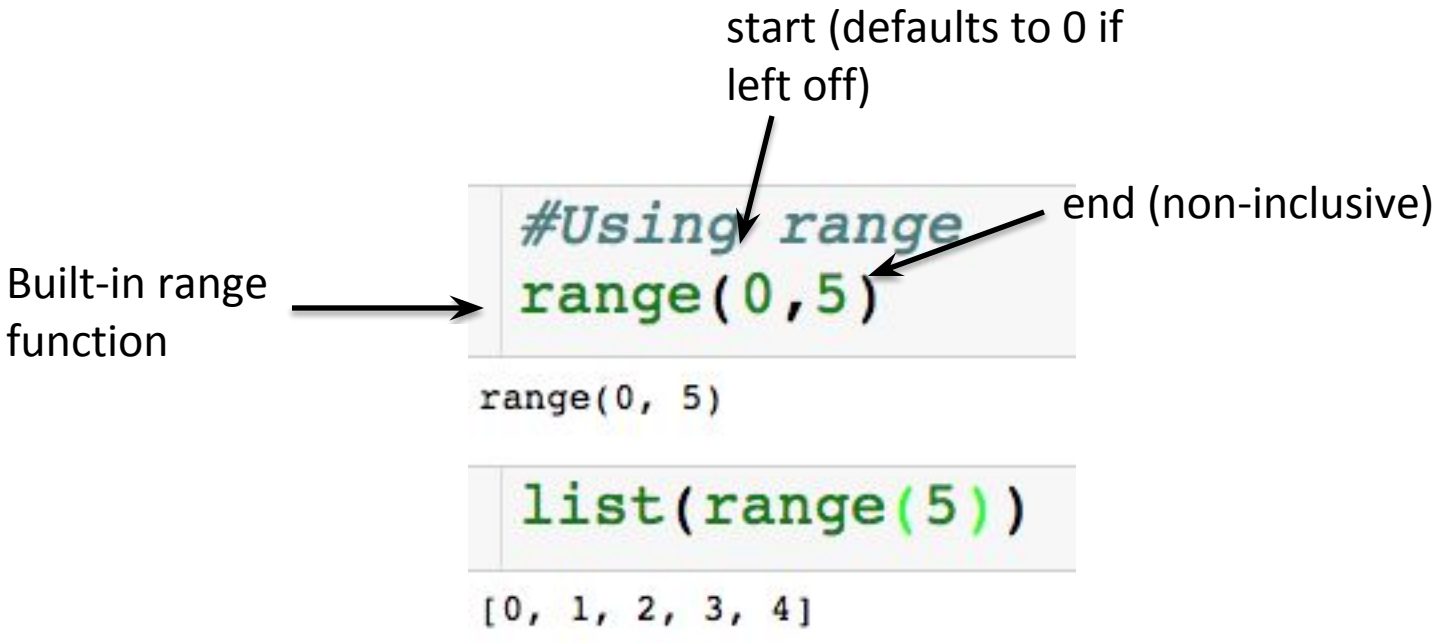
Summary: For Loops w/ Strings or Lists

- The loop target can be a string or a list
  - If the loop target is a list – iterate over the elements of the list by increasing index
  - If the loop target is a string – iterate over the characters of the string by increasing index
- We will see other python objects that can be loop for targets

Range


start (defaults to 0 if left off)
end (non-inclusive)
Built-in range function
#Using range
range(0,5)
range(0, 5)
list(range(5))
[0, 1, 2, 3, 4]

Range



- Can use range(0,n) as loop target
  - Need consecutive integers
  - Need to repeat something n times

For Loops w/ Range

- Using range() as loop target to get consecutive integers

- Compute even numbers >=0

```
even_nums = []
for i in range(5):
    if i%2==0:
        even_nums+=[i]
```

For Loops w/ Range

$$\text{even nums} = [\,]$$
$$\text{i} = 0$$

```
even_nums = []
for i in range(5):
    if i%2==0:
        even_nums+=[i]
```

For Loops w/ Range

$$\text{even nums} = [\,]$$
$$\text{i} = 0$$

```
even_nums = []
for i in range(5):
    if i%2==0:
        even_nums+=[i]
```

For Loops w/ Range

$$\text{even\_nums} = [0]$$
$$\text{i} = 0$$

```
even_nums = []
for i in range(5):
    if i%2==0:
        even_nums+=[i]
```

For Loops w/ Range

$$\text{even nums} = [0]$$
$$\text{i} = 1$$

```
even_nums = []
for i in range(5):
    if i%2==0:
        even_nums+=[i]
```

For Loops w/ Range

$$\text{even nums} = [0]$$
$$\text{i} = 1$$

```
even_nums = []
for i in range(5):
    if i%2==0:
        even_nums+=[i]
```

For Loops w/ Range

$$\text{even nums} = [0]$$

$$\text{i} = 2$$

```
even_nums = []
for i in range(5):
    if i%2==0:
        even_nums+=[i]
```

For Loops w/ Range

$$\text{even\_nums} = [0]$$
$$\text{i} = 2$$

```
even_nums = []
for i in range(5):
    if i%2==0:
        even_nums+=[i]
```

For Loops w/ Range

$$\text{even\_nums} = [0, 2]$$
$$\text{i} = 2$$

```
even_nums = []
for i in range(5):
    if i%2==0:
        even_nums+=[i]
```

For Loops w/ Range

$$\text{even nums} = [0, 2]$$
$$\text{i} = 3$$

```
even_nums = []
for i in range(5):
    if i%2==0:
        even_nums+=[i]
```

And so on...

For Loops w/ Range

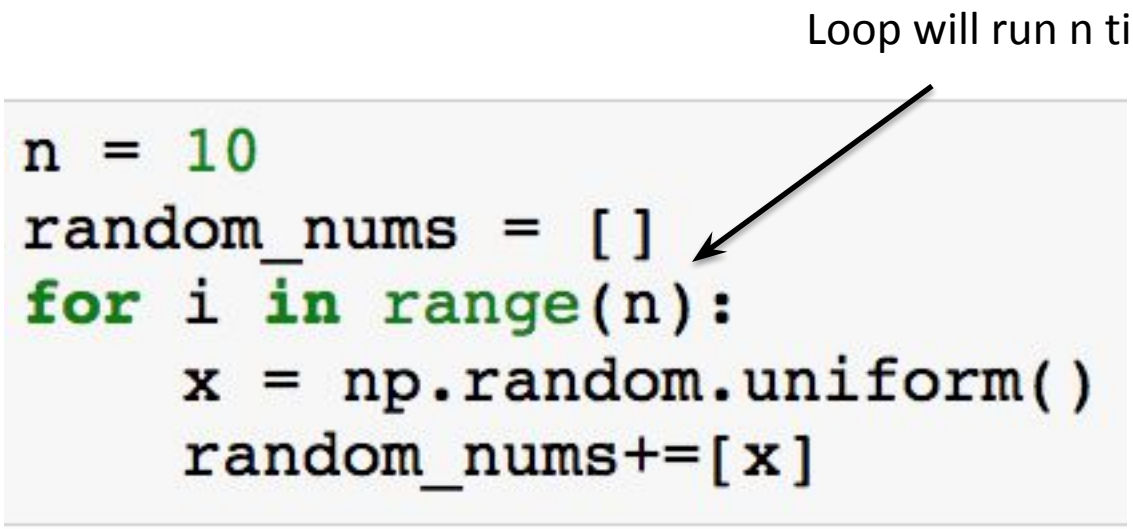
- Using range() as loop target to do something n times.
  - Loop variables will not be used

- Get n random samples of U[0,1]

For Loops w/ Range

- Using range() as loop target to do something n times.
  - Loop variables will not be used

- Get n random samples of U[0,1]



Break Statement

- Jump out of nearest loop
- Check if x is prime – just need to find one factor

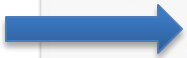
```
x= 15
flag=0
for i in range(2,x):
    if x%i ==0:
        flag=1
        mult = i
        break

if flag==0:
    print("%d is prime" %x)
else:
    print("%d is multiple of %d" %(mult, x))
```

Break Statement

x=15

flag=0

i = 2



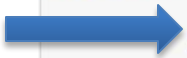
```
x= 15
flag=0
for i in range(2,x):
    if x%i ==0:
        flag=1
        mult = i
        break

if flag==0:
    print("%d is prime" %x)
else:
    print("%d is multiple of %d" %(mult, x))
```

Break Statement

x=15

flag=0

i = 2



```
x= 15
flag=0
for i in range(2,x):
    if x%i ==0:
        flag=1
        mult = i
        break

if flag==0:
    print("%d is prime" %x)
else:
    print("%d is multiple of %d" %(mult, x))
```

Break Statement

x=15

flag=0

i = 3

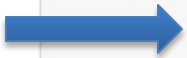
```
x= 15
flag=0
for i in range(2,x):
    if x%i ==0:
        flag=1
        mult = i
        break

if flag==0:
    print("%d is prime" %x)
else:
    print("%d is multiple of %d" %(mult, x))
```

Break Statement

x=15

flag=0

i = 3



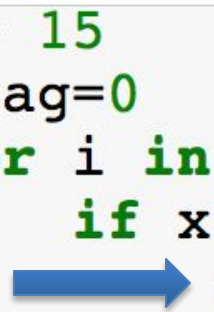
```
x= 15
flag=0
for i in range(2,x):
    if x%i ==0:
        flag=1
        mult = i
        break

if flag==0:
    print("%d is prime" %x)
else:
    print("%d is multiple of %d" %(mult, x))
```

Break Statement

x=15

flag=1

i = 3



```
x= 15
flag=0
for i in range(2,x):
    if x%i ==0:
        flag=1
        mult = i
        break

if flag==0:
    print("%d is prime" %x)
else:
    print("%d is multiple of %d" %(mult, x))
```

Break Statement

x=15

flag=1

i = 3

mult = 3

```
x= 15
flag=0
for i in range(2,x):
    if x%i ==0:
        flag=1
        mult = i
        break

if flag==0:
    print("%d is prime" %x)
else:
    print("%d is multiple of %d" %(mult, x))
```

Break Statement

x=15

flag=1

i = 3

mult = 3

```
x= 15
flag=0
for i in range(2,x):
    if x%i ==0:
        flag=1
        mult = i
        break

if flag==0:
    print("%d is prime" %x)
else:
    print("%d is multiple of %d" %(mult, x))
```

Break Statement

x=15

flag=1

i = 3

mult = 3

```
x= 15
flag=0
for i in range(2,x):
    if x%i ==0:
        flag=1
        mult = i
        break

if flag==0:
    print("%d is prime" %x)
else:
    print("%d is multiple of %d" %(mult, x))
```

Break Statement

x=15

flag=1

i = 3

mult = 3

```
x= 15
flag=0
for i in range(2,x):
    if x%i ==0:
        flag=1
        mult = i
        break

if flag==0:
    print("%d is prime" %x)
else:
    print("%d is multiple of %d" %(mult, x))
```

Break Statement

$x=15$

$flag=1$

$i = 3$

$mult = 3$

```
x= 15
flag=0
for i in range(2,x):
    if x%i ==0:
        flag=1
        mult = i
        break

if flag==0:
    print("%d is prime" %x)
else:
    print("%d is multiple of %d" %(mult, x))
```

Output: “3 is multiple of 15”