

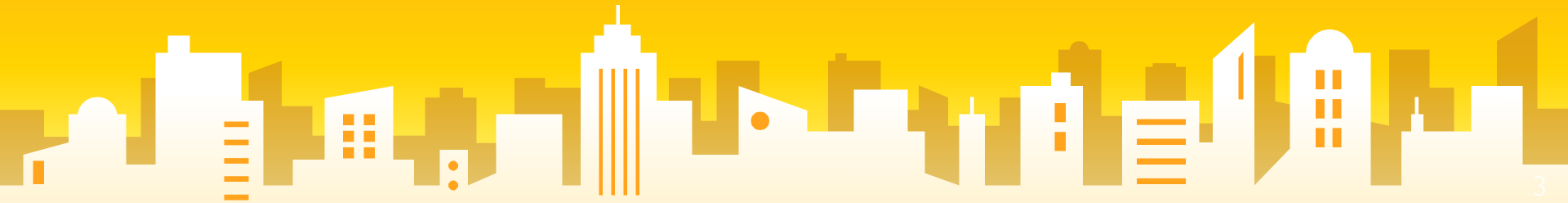
Welcome Back



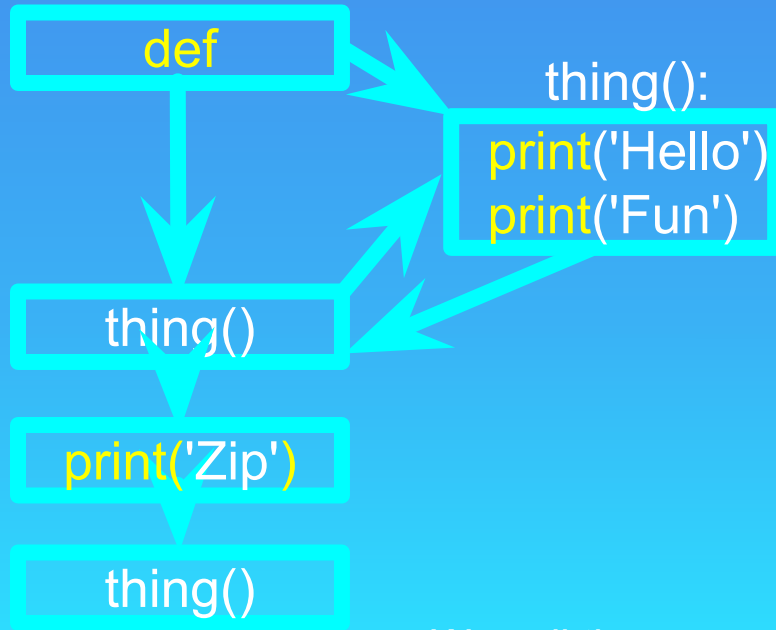


Functions in Python

First Lesson.



Stored (and reused) Steps



Program:

```
def thing():  
    print('Hello')  
    print('Fun')
```

```
thing()  
print('Zip')  
thing()
```

Output:

```
Hello  
Fun  
Zip  
Hello  
Fun
```

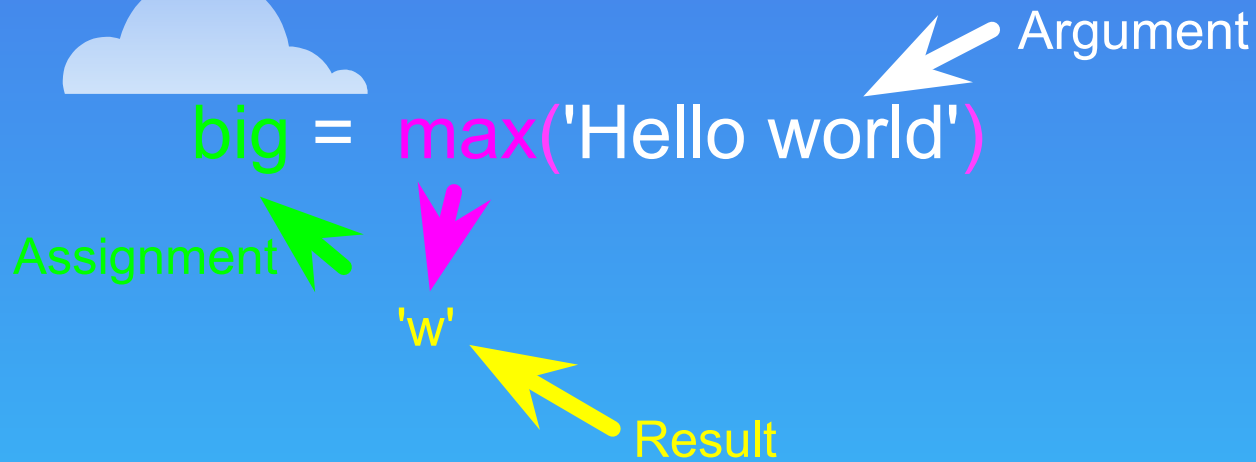
We call these reusable pieces of code “functions”

Python Functions

- There are two kinds of functions in Python.
 - **Built-in functions** that are provided as part of Python - `print()`, `input()`, `type()`, `float()`, `int()` ...
 - **Functions that we define ourselves** and then use
- We treat the built-in function names as “new” **reserved words**
(i.e., we avoid them as variable names)

Function Definition

- In Python a **function** is some reusable code that takes **arguments**(s) as input, does some computation, and then returns a result or results
- We define a **function** using the **def** reserved word
- We call/invoke the **function** by using the function name, parentheses, and **arguments** in an expression



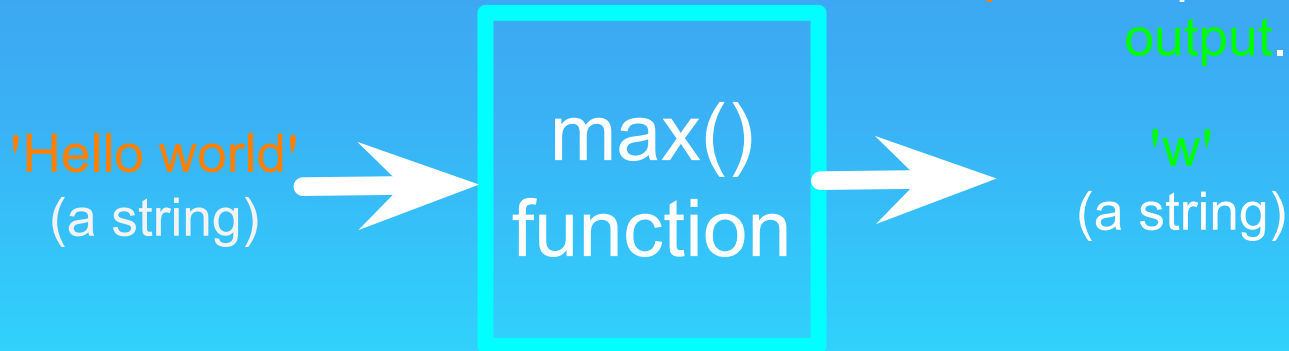
```
>>> big = max('Hello world')
>>> print(big)
w
>>> tiny = min('Hello world')
>>> print(tiny)

>>>
```

Max Function

```
>>> big = max('Hello world')  
>>> print(big)  
w
```

A function is some
stored code that we use.
A function takes some
input and produces an
output.



Guido wrote this code

Max Function

```
>>> big = max('Hello world')
>>> print(big)
w
```

A function is some
stored code that we use.
A function takes some
input and produces an
output.

'Hello world'
(a string)



```
def max(inp):
    blah
    blah
    for x in
inp:
    blah
    blah
```



'w'
(a string)

Guido wrote this code

Type Conversions

- When you put an integer and floating point in an expression, the integer is **implicitly** converted to a float
- You can control this with the built-in functions `int()` and `float()`

```
>>> print(float(99) / 100)
0.99
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class 'float'>
>>> print(1 + 2 * float(3) / 4 - 5)
-2.5
>>>
```

String Conversions

- You can also use `int()` and `float()` to convert between strings and integers
- You will get an **error** if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str'
and 'int'
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsval = 'hello bob'
>>> niv = int(nsval)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```

Building our Own Functions

- We create a new function using the **def** keyword followed by optional parameters in parentheses
- We indent the body of the function
- This **defines** the function but **does not** execute the body of the function

```
def print_lyrics():  
    print("I'm a lumberjack, and I'm okay.")  
    print('I sleep all night and I work all  
day.')
```

```
x = 5  
print('Hello')
```

```
def print_lyrics():  
    print("I'm a lumberjack, and I'm okay.")  
    print('I sleep all night and I work all  
day.')
```

```
print('Yo')  
x = x + 2  
print(x)
```

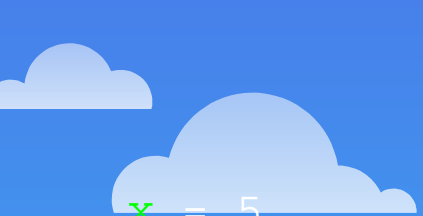
`print_lyrics():`

```
print("I'm a lumberjack, and I'm okay.")  
print('I sleep all night and I work all day.')
```

Hello
Yo
7

Definitions and Uses

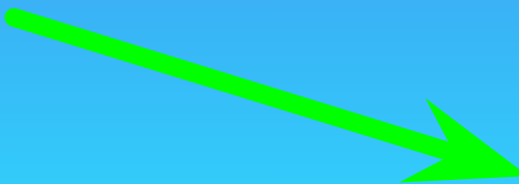
- Once we have **defined** a function, we can **call** (or **invoke**) it as many times as we like
- This is the **store** and **reuse** pattern



```
x = 5  
print('Hello')
```

```
def print_lyrics():  
    print("I'm a lumberjack, and I'm okay.")  
    print('I sleep all night and I work all day.')
```

```
print('Yo')  
print_lyrics()  
x = x + 2  
print(x)
```



Hello
Yo
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
7

Arguments

- An **argument** is a value we pass into the **function** as its **input** when we call the function
- We use **arguments** so we can direct the **function** to do different kinds of work when we call it at **different** times
- We put the **arguments** in parentheses after the **name** of the function

```
big = max('Hello world')
```



Argument

Parameters

A **parameter** is a variable which we use **in** the function **definition**. It is a “handle” that allows the code in the function to access the **arguments** for a particular function invocation.

```
>>> def greet(lang):  
...     if lang == 'es':  
...         print('Hola')  
...     elif lang == 'fr':  
...         print('Bonjour')  
...     else:  
...         print('Hello')  
...  
>>> greet('en')  
Hello  
>>> greet('es')  
Hola  
>>> greet('fr')  
Bonjour  
>>>
```


Return Values

Often a function will take its arguments, do some computation, and **return** a value to be used as the value of the function call in the **calling expression**. The **return** keyword is used for this.

```
def greet():  
    return "Hello"
```

```
print(greet(), "Glenn")  
print(greet(), "Sally")
```

```
Hello Glenn  
Hello Sally
```

Return Value

- A “fruitful” **function** is one that produces a **result** (or **return value**)
- The **return** statement ends the **function** execution and “sends back” the **result** of the **function**

```
>>> def greet(lang):  
...     if lang == 'es':  
...         return 'Hola'  
...     elif lang == 'fr':  
...         return 'Bonjour'  
...     else:  
...         return 'Hello'  
...  
>>> print(greet('en'), 'Glenn')  
Hello Glenn  
>>> print(greet('es'), 'Sally')  
Hola Sally  
>>> print(greet('fr'), 'Michael')  
Bonjour Michael  
>>>
```

Arguments, Parameters, and Results

```
>>> big = max('Hello world')  
>>> print(big)  
w
```

Argument → 'Hello world'

Parameter

```
def max(inp):  
    blah  
    blah  
    for x in  
inp:  
    blah  
    blah  
    return 'w'
```

→ 'w'
Result

Multiple Parameters / Arguments

- We can define more than one **parameter** in the **function definition**
- We simply add more **arguments** when we call the **function**
- We match the number and order of arguments and parameters

```
def addtwo(a, b):  
    added = a + b  
    return added
```

```
x = addtwo(3, 5)  
print(x)
```

8

Void (non-fruitful) Functions

- When a function does not return a value, we call it a “void” function
- Functions that return values are “fruitful” functions
- Void functions are “not fruitful”

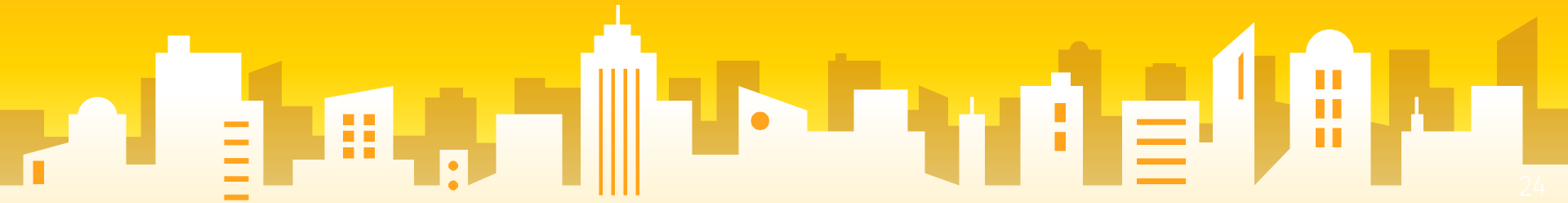
To function or not to function...

- Organize your code into “paragraphs” - capture a complete thought and “name it”
- Don’t repeat yourself - make it work once and then reuse it
- If something gets too long or complex, break it up into logical chunks and put those chunks in functions
- Make a library of common stuff that you do over and over - perhaps share this with your friends...



Write a Python function to find the Max
of three numbers.

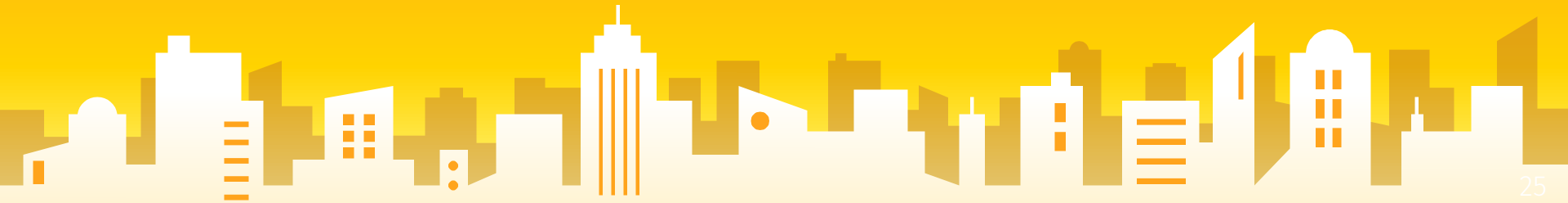
Assignment





Write a Python function to sum all the
numbers in a list.

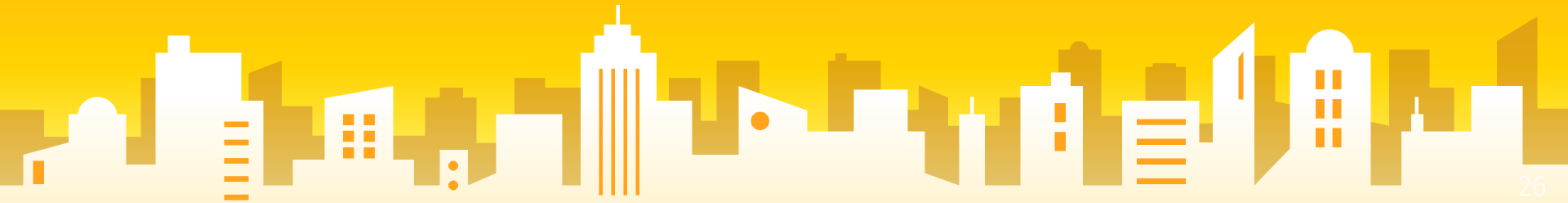
Assignment





Write a Python function to check
whether a number is in a given range.

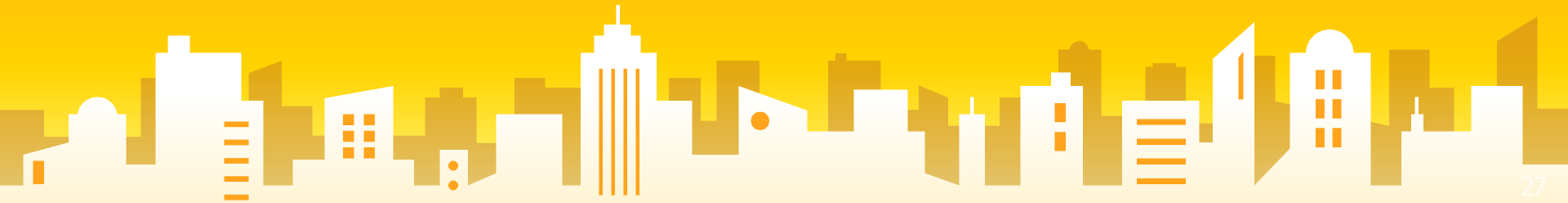
Assignment






Write a Python program to print the
even numbers from a given list.

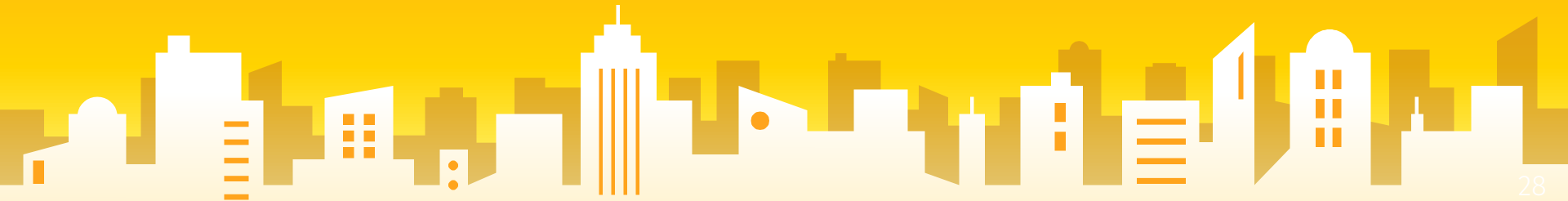
Assignment





Write a Python function that checks
whether a passed string is palindrome
or not.

Assignment



**Missing
Me**

Strings

Second Lesson



String Data Type

- A string is a sequence of characters
- A string literal uses quotes
'Hello' or "Hello"
- For strings, + means “concatenate”
- When a string contains numbers, it is still a string
- We can convert numbers in a string into a number using `int()`

```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print(bob)
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> x = int(str3) + 1
>>> print(x)
124
>>>
```

Reading and Converting

- We prefer to read data in using **strings** and then parse and convert the data as we need
- This gives us more control over error situations and/or bad user input
- Input numbers must be **converted** from strings

```
>>> name = input('Enter:')
Enter:Chuck
>>> print(name)
Chuck
>>> apple = input('Enter:')
Enter:100
>>> x = apple - 10
Traceback (most recent call
last):  File "<stdin>", line
1, in <module>
TypeError: unsupported operand
type(s) for -: 'str' and 'int'
>>> x = int(apple) - 10
>>> print(x)
90
```



Looking Inside Strings

- We can get at any single character in a string using an index specified in **square brackets**
- The index value must be an integer and starts at zero
- The index value can be an expression that is computed

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

A Character Too Far

- You will get a **python error** if you attempt to index beyond the end of a string
- So be careful when constructing index values and slices

```
>>> zot = 'abc'
>>> print(zot[5])
Traceback (most recent call
last):  File "<stdin>", line
1, in <module>
IndexError: string index out
of range
>>>
```


Strings Have Length

The built-in function `len` gives us the length of a string

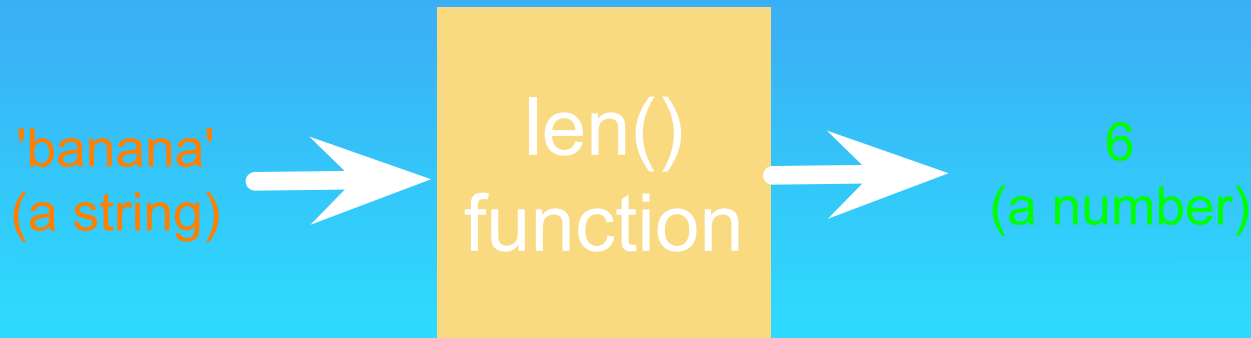


```
>>> fruit = 'banana'
>>> print(len(fruit))
6
```

len Function

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print(x)
6
```

A **function** is **some stored code** that we use. A function takes some **input** and produces an **output**.



len Function

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print(x)
6
```

A **function** is **some stored code** that we use. A function takes some **input** and produces an **output**.

'banana'
(a string)



```
def len(inp):
    blah
    blah
    for x in
y:
    blah
    blah
```



6
(a number)

Looping Through Strings

Using a **while** statement, an **iteration variable**, and the **len** function, we can construct a loop to look at each of the letters in a string individually

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

0 b
1 a
2 n
3 a
4 n
5 a

Looping Through Strings

- A definite loop using a **for** statement is much more elegant
- The **iteration variable** is completely taken care of by the **for** loop

```
fruit = 'banana'  
for letter in fruit:  
    print(letter)
```

b
a
n
a
n
a

Looping Through Strings

- A definite loop using a **for** statement is much more **elegant**
- The **iteration variable** is completely taken care of by the **for** loop

```
fruit = 'banana'
for letter in fruit :
    print(letter)
```

```
index = 0
while index < len(fruit)
:
    letter = fruit[index]
    print(letter)
    index = index + 1
```

b
a
n
a
n
a

Looping and Counting


This is a simple loop that loops through each letter in a string and counts the number of times the loop encounters the 'a' character

```
word = 'banana'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print(count)
```

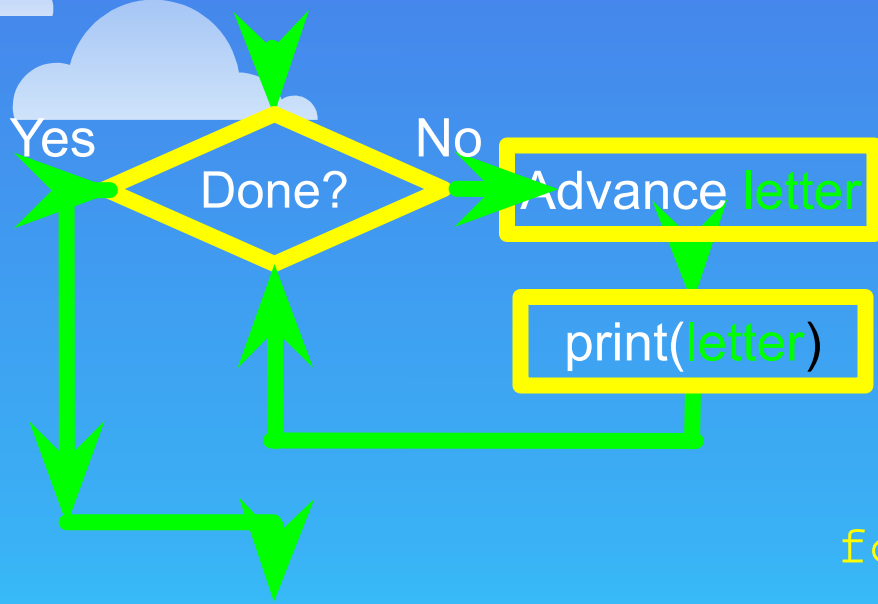
Looking Deeper into in

- The **iteration variable** “iterates” through the **sequence** (ordered set)
- The **block (body)** of code is executed once for each value **in** the **sequence**
- The **iteration variable** moves through all of the values **in** the **sequence**

Iteration variable **Six-character string**



```
for letter in 'banana' :  
    print(letter)
```

```
for letter in 'banana' :  
    print(letter)
```

The **iteration variable** “iterates” through the **string** and the **block (body)** of code is executed once for each value **in** the **sequence**

Slicing Strings

- We can also look at any continuous section of a string using a **colon operator**
- The second number is one beyond the end of the slice - “up to but not including”
- If the second number is beyond the end of the string, it stops at the end

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
```

Slicing Strings

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

If we leave off the first number or the last number of the slice, it is assumed to be the beginning or end of the string respectively

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```

String Concatenation

When the `+` operator is applied to strings, it means “concatenation”

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print(b)
HelloThere
>>> c = a + ' ' + 'There'
>>> print(c)
Hello There
>>>
```

Using **in** as a Logical Operator

- The **in** keyword can also be used to check to see if one string is “in” another string
- The **in** expression is a logical expression that returns **True** or **False** and can be used in an **if** statement

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Found it!')
...
Found it!
>>>
```

String Comparison

```
if word == 'banana':  
    print('All right, bananas.')  
if word < 'banana':  
    print('Your word,' + word + ', comes before banana.')elif word > 'banana':  
    print('Your word,' + word + ', comes after banana.')else:  
    print('All right, bananas.')
```

String Library

- Python has a number of string **functions** which are in the **string library**
- These **functions** are already **built into** every string - we invoke them by appending the function to the string variable
- These **functions** do not modify the original string, instead they return a new string that has been altered

```
>>> greet = 'Hello Bob'
>>> zap = greet.lower()
>>> print(zap)
hello bob
>>> print(greet)
Hello Bob
>>> print('Hi There'.lower())
hi there
>>>
```

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
```

<https://docs.python.org/3/library/stdtypes.html#string-methods>

str.replace(*old*, *new*[, *count*])

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

str.rfind(*sub*[, *start*[, *end*]])

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within *s*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

str.rindex(*sub*[, *start*[, *end*]])

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

str.rjust(*width*[, *fillchar*])

Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

str.rpartition(*sep*)

Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

str.rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done, the *rightmost* ones. If *sep* is not specified or `None`, any whitespace string is a separator. Except for splitting from the right, `rsplit()` behaves like `split()` which is described in detail below.

String Library

```
str.capitalize()  
str.center(width[, fillchar])  
str.endswith(suffix[, start[, end]])  
str.find(sub[, start[, end]])  
str.lstrip([chars])
```

```
str.replace(old, new[, count])  
str.lower()  
str.rstrip([chars])  
str.strip([chars])  
str.upper()
```

Searching a String

- We use the `find()` function to search for a substring within another string
- `find()` finds the first occurrence of the substring
- If the substring is not found, `find()` returns `-1`
- Remember that string position starts at zero



```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

Making everything UPPER CASE

- You can make a copy of a string in **lower case** or **upper case**
- Often when we are searching for a string using **find()** we first convert the string to lower case so we can search a string regardless of case

```
>>> greet = 'Hello Bob'
>>> nnn = greet.upper()
>>> print(nnn)
HELLO BOB
>>> www = greet.lower()
>>> print(www)
hello bob
>>>
```

Search and Replace

- The `replace()` function is like a “search and replace” operation in a word processor

```
>>> greet = 'Hello Bob'
>>> nstr = greet.replace('Bob', 'Jane')
>>> print(nstr)
Hello Jane
>>> nstr = greet.replace('o', 'X')
>>> print(nstr)
HellX Bxb
>>>
```

- It replaces **all** occurrences of the search string with the replacement string

Stripping Whitespace

- Sometimes we want to take a string and remove whitespace at the beginning and/or end
- `lstrip()` and `rstrip()` remove whitespace at the left or right
- `strip()` removes both beginning and ending whitespace

```
>>> greet = '    Hello Bob    '  
>>> greet.lstrip()  
'Hello Bob    '  
>>> greet.rstrip()  
'    Hello Bob'  
>>> greet.strip()  
'Hello Bob'  
>>>
```

Prefixes

```
>>> line = 'Please have a nice day'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

Parsing and Extracting

21

31

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ', atpos)
>>> print(sppos)
31
>>> host = data[atpos+1 : sppos]
>>> print(host)
uct.ac.za
```



Two Kinds of Strings

Python 2.7.10

```
>>> x = '이광춘'
>>> type(x)
<type 'str'>
>>> x = u'이광춘'
>>> type(x)
<type 'unicode'>
>>>
```

Python 3.5.1

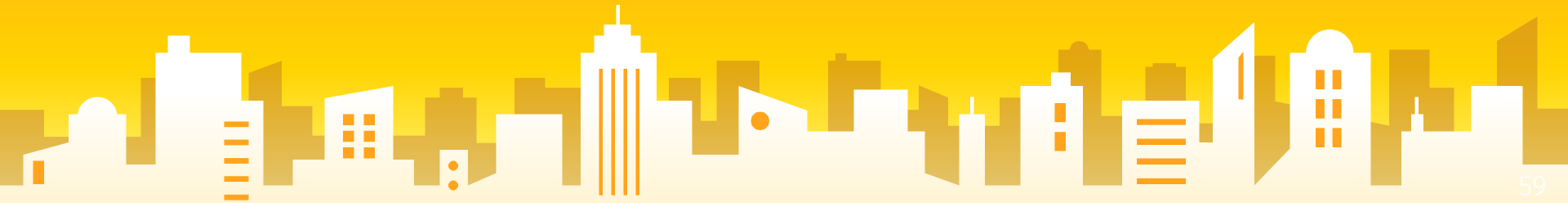
```
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
>>>
```


In Python 3, all strings are Unicode



Write a Python program to calculate the
length of a string.

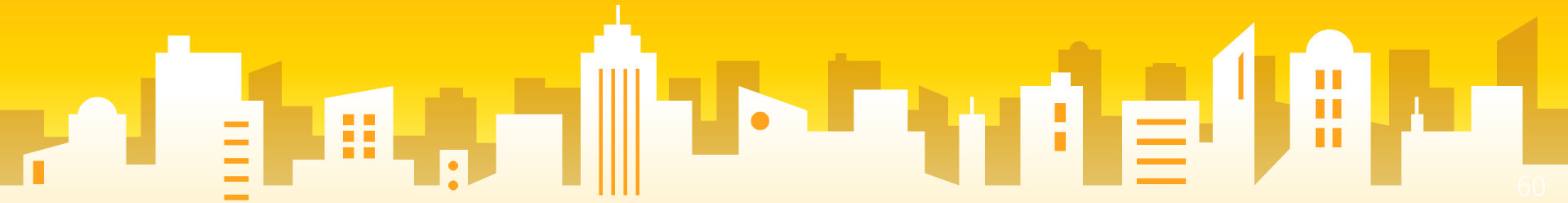
Assignment






Write a Python program to count the
number of characters (character
frequency) in a string.

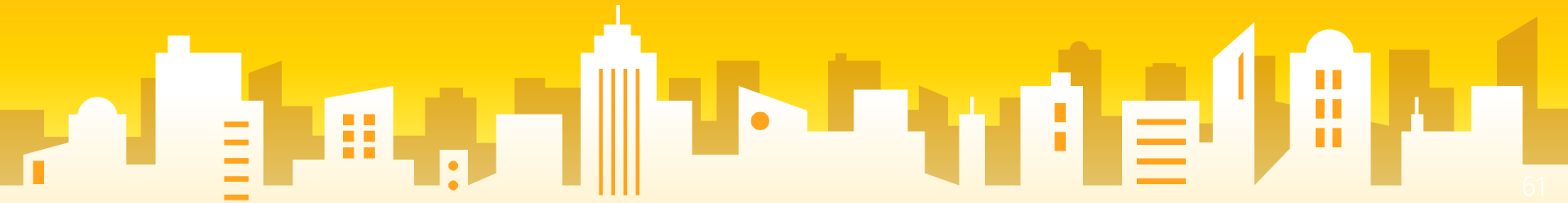
Assignment






Write a Python program to get a single string from two given strings, separated by a space and swap the first two characters of each string.

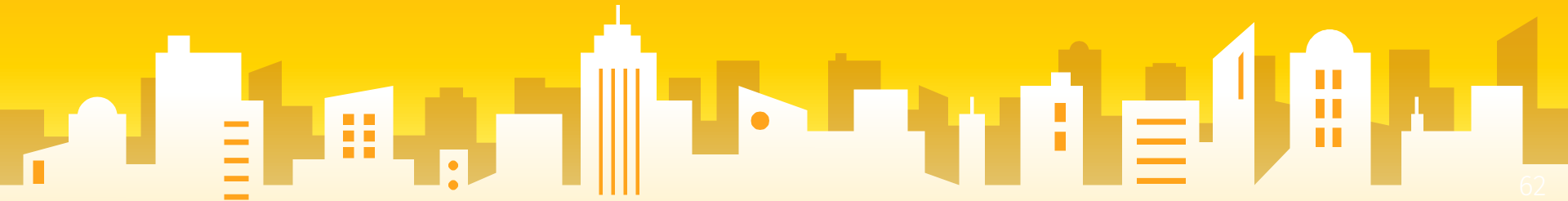
Assignment

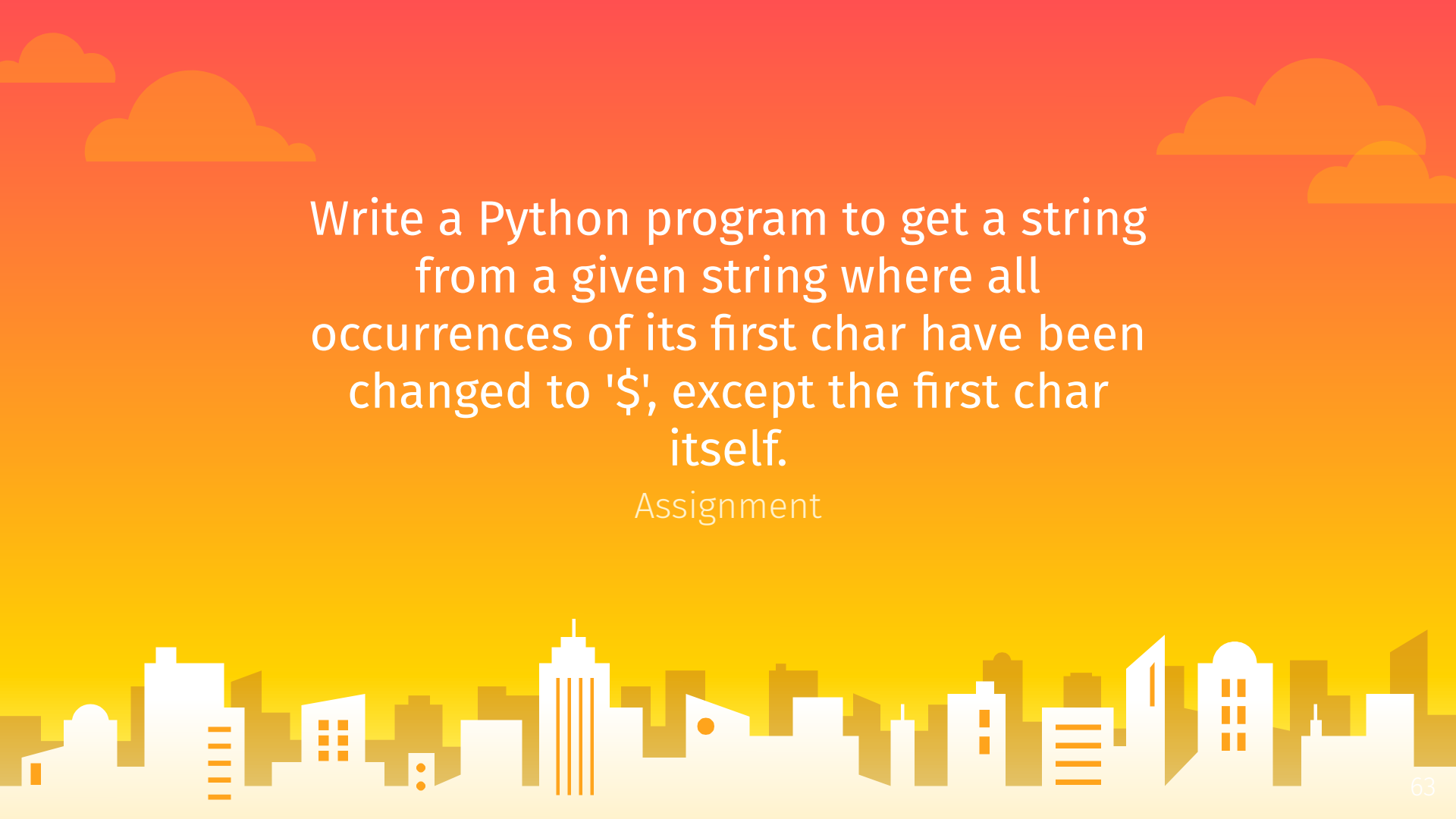




Write a Python program to add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged.

Assignment



The background of the slide features a stylized city skyline at the bottom with various white and yellow buildings. The sky is a gradient from orange at the top to yellow at the bottom, with several orange and yellow clouds scattered across it.

Write a Python program to get a string
from a given string where all
occurrences of its first char have been
changed to '\$', except the first char
itself.

Assignment

Thank you
Miss You...

