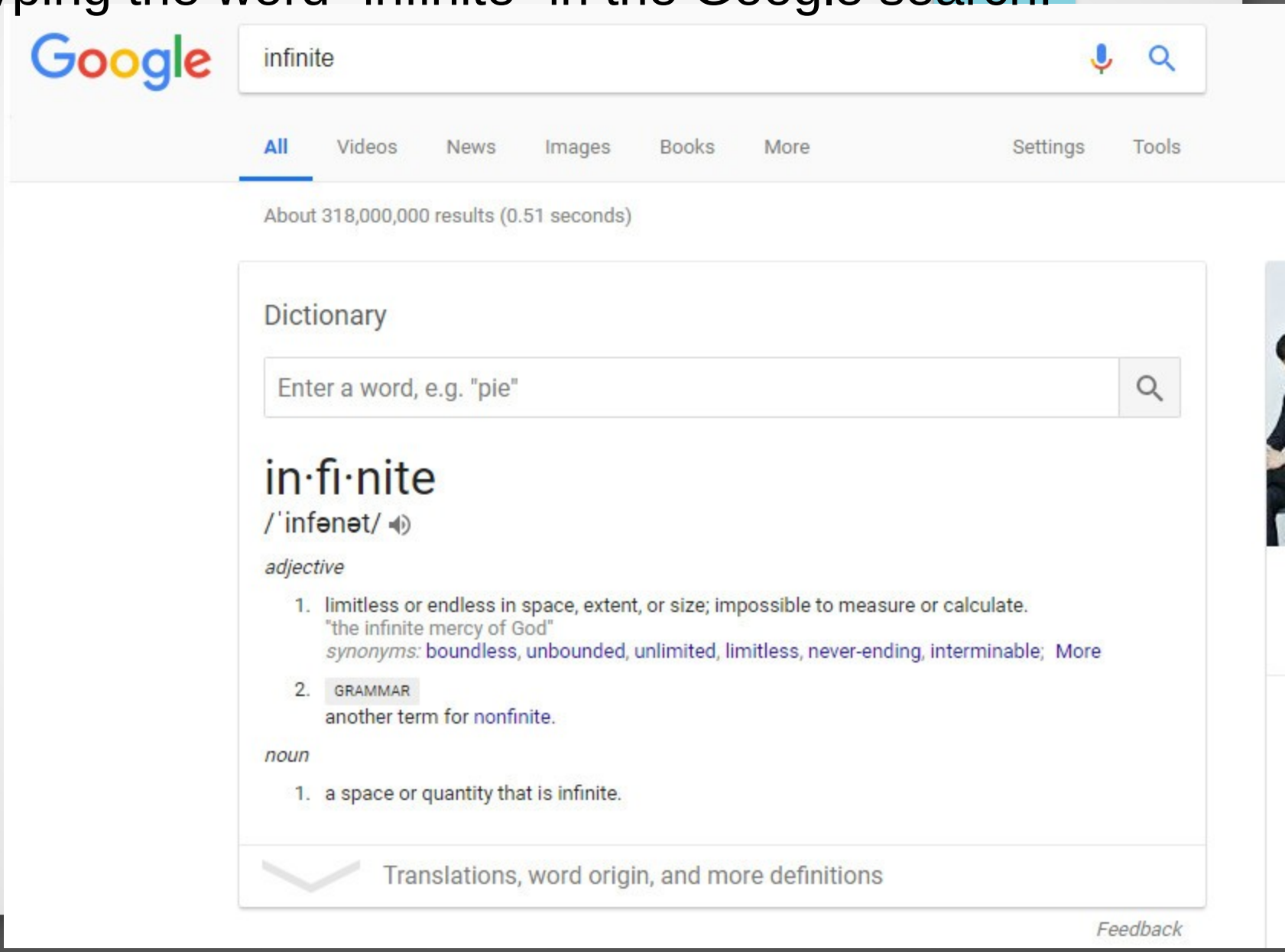
# Lecture 9

Topics to be covered:

- Dictionary basics
- Common data types summary
- String formatting

# Dictionary basics

Consider typing the word “infinite” in the Google search:



# Dictionary basics

A *dictionary* is a Python *container* used to describe associative relationships.

A *dictionary* is represented by the **`dict`** object type.

A *dictionary associates* (or "*maps*") *keys* with *values*.

A *key* is a term that can be located in a *dictionary*, such as the word "infinite" in the Google search.

A *value* describes some data associated with a *key*, such as a definition.

A *key* can be any immutable type, such as a number, string, or tuple; a *value* can be any type.

# Dictionary basics

A **`dict`** object is created using curly braces `{}` to surround the `key:value` pairs that comprise the dictionary contents.

**Example:**

```
myDict = {
   “street address”: ”2155 University Avenue”,
   “city”: “Bronx”,
   “state” : ”New York”,
   “zip code” : 10453,
   “phone” : “(718) 289-5100”,
   “admissions” : “(718) 289-5895”}
```

# Dictionary basics

Dictionaries are typically used in place of lists when an *associative relationship* exists.

**Example:** If a program contains a collection of anonymous student test scores, those scores should be stored in a list. However, if each score is associated with a student name, a dictionary could be used to associate student names to their score.

# Dictionary basics: in-class work

I have 5 students: Cute Princess, Fairy Queen, Evil Don, Fussy Cat, and Lazy Daisy.

I also have the record of 4 of their test scores.

Let's create a dictionary `students`, where student's IDs will serve as *`keys`*, and the *`value`* will be a *`list`* with five elements/members: student's name, and 4 test scores.

| Name | ID | Test 1 | Test 2 | Test 3 | Test 4 |
|---|---|---|---|---|---|
| Cute Princess | 846563 | 89 | 67 | 98 | 100 |
| Fairy Queen | 736542 | 76 | 56 | 83 | 99 |
| Evil Don | 287563 | 52 | 81 | 79 | 27 |
| Fussy Cat | 294512 | 27 | 38 | 100 | 75 |
| Lazy Daisy | 975321 | 88 | 99 | 66 | 77 |

Download and save file Dict1.py from our web-site

# Dictionary basics: in-class work

`students` dictionary we got:

| keys |
|---|
| 846563 |
| 736542 |
| 287563 |
| 294512 |
| 975321 |

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| “Cute Princess” | 89 | 67 | 98 | 100 |
| “Fairy Queen” | 76 | 56 | 83 | 99 |
| “Evil Don” | 52 | 81 | 79 | 27 |
| “Fussy Cat” | 27 | 38 | 100 | 75 |
| “Lazy Daisy” | 88 | 99 | 66 | 77 |

# Dictionary basics: in-class work

Now, let's print some information: put the following lines into *Dict1.py:*

```
print(students[975321])
print(students[846563])
```

See what happens!

| keys | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 846563 | "Cute Princess" | 89 | 67 | 98 | 100 |
| 736542 | "Fairy Queen" | 76 | 56 | 83 | 99 |
| 287563 | "Evil Don" | 52 | 81 | 79 | 27 |
| 294512 | "Fussy Cat" | 27 | 38 | 100 | 75 |
| 975321 | "Lazy Daisy" | 88 | 99 | 66 | 77 |

# Dictionary basics: in-class work

Let's now calculate Lazy Daisy's average test score: add the following lines of code into *Dict1.py*

```
s = students[975321]
averageTestScore = (s[1]+s[2]+s[3]+s[4])/4
print("Lazy Daisy average test score is",
   averageTestScore)
```

| keys | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 846563 | "Cute Princess" | 89 | 67 | 98 | 100 |
| 736542 | "Fairy Queen" | 76 | 56 | 83 | 99 |
| 287563 | "Evil Don" | 52 | 81 | 79 | 27 |
| 294512 | "Fussy Cat" | 27 | 38 | 100 | 75 |
| 975321 | "Lazy Daisy" | 88 | 99 | 66 | 77 |

s

s[0] s[1] s[2] s[3] s[4]

# Dictionary basics: in-class work

Now, let's add one more record and display the dictionary:

| Name | ID | Test 1 | Test 2 | Test 3 | Test 4 |
|---|---|---|---|---|---|
| "Glad Lad" | 625342 | 98 | 76 | 48 | 80 |

By adding the following line in *Dict1.py*:

```
students[625342] = ["Glad Lad",98,76,48,80]
print(students)
```

| keys | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 846563 | "Cute Princess" | 89 | 67 | 98 | 100 |
| 736542 | "Fairy Queen" | 76 | 56 | 83 | 99 |
| 287563 | "Evil Don" | 52 | 81 | 79 | 27 |
| 294512 | "Fussy Cat" | 27 | 38 | 100 | 75 |
| 975321 | "Lazy Daisy" | 88 | 99 | 66 | 77 |
| 625342 | "Glad Lad" | 98 | 76 | 48 | 80 |

# Dictionary basics: in-class work

Now, let's delete the record about Fussy Cat from the dictionary
By adding the following line in *Dict1.py*:

```
del students[294512]
print(students)
```

| keys | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 846563 | "Cute Princess" | 89 | 67 | 98 | 100 |
| 736542 | "Fairy Queen" | 76 | 56 | 83 | 99 |
| 287563 | "Evil Don" | 52 | 81 | 79 | 27 |
| ~~294512~~ | ~~"Fussy Cat"~~ | ~~27~~ | ~~38~~ | ~~100~~ | ~~75~~ |
| 975321 | "Lazy Daisy" | 88 | 99 | 66 | 77 |
| 625342 | "Glad Lad" | 98 | 76 | 48 | 80 |

# Dictionary basics: in-class work

Do the items 9-10 in the in-class handout

# Common data types summary

Numeric types `int` and `float` represent the most common types used to store data.

All numeric types support the normal mathematical operations such as addition, subtraction, multiplication, and division, among others.

| Type | Notes |
|---|---|
| int | Numeric type: Used for variable-width integers.<br>`0  9  -19  10286  -1937464` |
| float | Numeric type: Used for floating-point numbers.<br>`1.2  -8.876  1.00000008  -0.0000009` |

# Common data types summary

Sequence types `string`, `list`, and `tuple` are all containers for collections of objects *ordered by position in the sequence*.

The first object has an index of 0 and subsequent elements have indices 1, 2, etc.

A `list` and a `tuple` are very similar, except that a `list` is *mutable* and individual elements may be edited or removed. Conversely, a `tuple` is *immutable* and individual elements may not be edited or removed .

`Lists` and `tuples` can contain any type.
A `string` contains only single-characters.
`len()` and `[ ]` can be applied to any sequence type.

# Common data types summary

The only mapping type in Python is the `dict` type.

Like a sequence type, a `dict` serves as a container. However, each element of a dict is independent, having no special ordering or relation to other elements.

A dictionary uses `key-value` pairs to associate a `key` with a `value`.

# Common data types summary

## Choosing a container type

New programmers often struggle with choosing the types that best fit their needs.

In general:

- a programmer might use a `list` when data has an order, such as lines of text on a page.
- a programmer might use a `tuple` instead of a `list` if the contained data should not change.
- if order is not important, a programmer might use a `dictionary` to capture relationships between elements, such as student names and grades.

# Common data types summary

## Example: plotting a graph of a function

Assume I have a bunch of points that I want to use to plot a graph of a function.
Here is what I have for now, but I believe there will be more...
(-3,6), (-2,1), (-1,-2), (0,-3), (1,-2), (2,1)

What type should I use?

# Common data types summary

## Example: passwords

I’m building a web service where each user would need to login into their account in order to use the services.

How should I store the passwords? What type should use?

# Common data types summary

## Example: phone numbers

I’m going to store a phone number of a friend of mine.

What type should I use?

# Type conversions

## Conversion methods

Sometimes we need explicitly convert an item's type. It can be done by using the following conversion methods:

| Function | Notes | Can convert |
|---|---|---|
| int() | creates integers | int, float, strings w/ integers only |
| float() | creates floats | int, float, strings w/ integers or fractions |
| str() | creates strings | any |

# Type conversions : in-class work

Type the following in the Python interpreter:

```
>>> a = 12.876
>>> b = int(a)
>>> print(b)

What do you see?

>>> c = str(a)
>>> print(c)

What do you see?

>>> d = “15.6”
>>> e = float(d)
>>> print(e)
```

# String formatting

Program output commonly includes the value of variables as a part of the text.

**Example:** for the following code

```
num = 18
tum = 9.8
print("I have a number",num,"and a number",tum)
```

# String formatting

Program output commonly includes the value of variables as a part of the text.

**Example:** for the following code

```
num = 18
tum = 9.8
print("I have a number",num,"and a number",tum)
```

will produce:

```
I have a number 18 and a number 9.8
```

Note that we have to keep track of those double quotes (") and commas in the print statement.

# String formatting

Compare the following two `print` statements:

```
num = 18,tum = 9.8
```

```
print("I have a number",num,"and a number",tum)
```

```
print("I have a number %d and a number %f" %
(num,tum))
```

The first one produced:

```
I have a number 18 and a number 9.8
```

The second one produced:

```
I have a number 18 and a number 9.800000
```

# String formatting

A *string formatting expression* allows a programmer to create a string with *placeholders* that are replaced by the *value of variables*.
Such a placeholder is called a *conversion specifier*.
Different conversion specifiers are used to perform a conversion of the given variable value to a different type when creating the string.

# String formatting

A *string formatting expression* allows a programmer to create a string with *placeholders* that are replaced by the *value of variables*.
Such a placeholder is called a *conversion specifier*.
Different conversion specifiers are used to perform a conversion of the given variable value to a different type when creating the string.

Example:

```
num = 5.5
print("The integer part is %d" % num)
```

will yield:

```
The integer part is 5
```

# String formatting

**Example:** Consider the following code fragment

```
price = 119 # in dollars
discount = 30 # in percent %
print("A $%d jacket at %d%% discount is now
priced at %f" % (price,discount,price*0.7))
```

Produces the output:

```
A $119 jacket at 30% discount is now priced at
83.300000
```

# String formatting

**Example:** Consider the following code fragment

```
price = 119 # in dollars
discount = 30 # in percent %
print("A $%d jacket at %d%% discount is now
priced at %f" % (price,discount,price*0.7))
```

Produces the output:

```
A $119 jacket at 30% discount is now priced at
83.300000
```

# String formatting

**Example:** Consider the following code fragment

```
name1 = "Chris"
name2 = "John"
print("%s and %s are heading to the movies
tonight" % (name1,name2)
```

Produces the output:

```
Chris and John are heading to the movies
tonight
```

# Practice

Proceed to in-class activity for Day 9 in the handout