
# CS 1112: Introduction To Programming

## Introduction to Python Dictionaries

Dr. Nada Basit // basit[*at*]Virginia[*dot*]edu

Fall 2023

# Friendly Reminders

- Your ***safety*** and ***comfort*** is important!
  - If you choose to wear a mask you are welcome to do so
  - *We will interpret wearing a mask as being considerate and caring of others in the classroom (not that you are sick), and realize that some may choose to mask to remain distanced*
- Be an ***active*** participant in your learning!
  You're welcome and ***encouraged*** to ask questions during class!
- If you feel ***unwell***, or think you are, please stay home
  - *We will work with you!*
  - Get some rest ☺
  - View the recorded lectures – *please allow 24-48 hours to post*
  - *Contact us!*

# Announcements

- **Quiz 6** is due by 11:00pm on 10/23 (***tonight***)!
- **PA05** is due by 11:00pm on 10/27 (***note the change***)!
  - Submit on Gradescope
  - Submit the right kinds of files
  - Submit files using the correct names
  - REMEMBER on Gradescope: you can submit an **UNLIMITED** number of times prior to the deadline. Look at the score you got, if you have some points taken off, that's ok, go back and fix your code and resubmit! Do this as often as you like BEFORE the assignment deadline. You cannot resubmit after the deadline.
  - ***REMEMBER ALSO***: You have a grace period of 24 hours to submit your PAs!
- **Exam 1**: feel free to schedule one-on-one time with me to review your exam, if you like!

# Properties of some collections

| **Type** | **Stores** | **Syntax** |
|---|---|---|
| Range | ints | `range(3,7)` |
| String | characters | `"Hello", "abc 123"` |
| List | anything | `[1,2,3,6,"hello"]` |
| Tuple | anything | `(1,2,3,6,"hello")` |
| Dictionary | key:value pairs | `{17:"hi", 29:"bye"}` |
| Set | anything | `{1,2,6,"hi"}` |

Today

# Kinds of Collections (the word "collection" in python technically has a much more strict meaning)

**Sequence types**

```
str    # string
range
list
tuple
```

- Order Matters
- Repetition of Items OK
- Counting starts at 0
- Collection[index] gives a specific value from the collection

**Non-Sequence types**

```
dict  # dictionary
set
```

- Ordering - not really?
  - first item is not at index 0
  - dict - insertion order only
  - set - no ordering
- No Repetition of Items Allowed

# Kinds of Collections (the word "collection" in python technically has a much more strict meaning)

**Sequence types**

```
str     # string
range
list
tuple
```

- Order Matters
- Repetition of Items OK
- Counting starts at 0
- Collection[index] gives a specific value from the collection

**Non-Sequence types**

```
dict   # dictionary
set
```

Today

- Ordering - not really?
  - first item is not at index 0
  - dict - insertion order only
  - set - no ordering
- No Repetition of Items Allowed

# Dictionaries (*Python uses the keyword:* `dict`)

- Like a list, but with index names that you create (called "KEYS")
- Each key is paired with a "VALUE"
- We can think of a dictionary similar to a list, but instead of indices 0, 1, 2, 3, 4, …, we choose the index (an int, or a string, …)
- Using a dictionary:

```
d = {}  # an empty dictionary named d

d = {4: "San Francisco", 7: "Edinburgh"} # 2 key-value pairs

d[12] = "apple"  # Adding a new key-value pair to a dictionary

x = d[4]        # Retrieving a value from a dictionary.

                # X will be assigned "hi"
```

# Lists vs. Dictionaries

## LIST

- Index to access members
- Indexes start with 0
- Indexes are consecutive ints
- To **add** a new thing: `list.append(something)`

## DICTIONARY

- Has keys to access members
- Each key must be unique
- Key can be:
  - Strings, ints, floats, booleans, tuples
  - (Not: lists, sets, dictionaries)
- To **add** a new thing: `d[key]=value`

# A dictionary contains **Key-Value Pairs**

- Think of **key-value pairs** like safety deposit boxes at a bank
- The ***values*** are stored in safety deposit boxes
- In order to access a value, you need the ***key*** to unlock the box
- Every box has a unique key

# Using a Dictionary

```
d = {}  # An empty dictionary named d

# A dictionary with 2 key-value pairs. The keys here are integers, the values are strings.
d = {4: "hi", 7: "bye"}

# A dictionary with 2 key-value pairs. The keys are different types.
d = {3: "banana", "Scores": [3,6,7] }


# Adding/Modifying a key-value pair
d[12] = "apple"   # Added a new key-value pair
d[12] = "pumpkin" # Modified the value of an existing key

# Retrieve a value from a dictionary. x will be assigned "pumpkin".
# Nothing was removed from d
x = d[12]


# Deleting a dictionary key-value pair
del d[12]

# How many key-value pairs in the dictionary
len(d)
```

# Some dictionary functions/methods

***copy()*** *– creates a* ***copy*** *of the dictionary*

```
p = orders.copy()
```

***keys()*** *- returns iterator(sequence) to the* ***set of key values*** *in the dictionary*

```
for person in orders.keys():
```

***values()*** *– returns iterator to the* ***set of values*** *in the dictionary*

```
for burger in orders.values():
```

***items()*** *– returns iterator to the* ***set of <key-value> pairs*** *in the dictionary*

```
for pair in orders.items():
```

# Looping through things

We can think of a dictionary as an unsorted list of key-value pairs

**List:**

Loop through a **list** by the **items** -

```
for my_item in my_list:
  Do stuff with my_item
```

Loop through a **list** by **index** -

```
for i in range(len(my_list)):
  my_item = my_list[i]
  Do stuff with my_item
```

**Dict:**

Loop through a dictionary -

```
for my_key in my_dictionary:
  Do stuff with my_key
  Do stuff with my_dictionary[my_key]
```

Can also loop through -

- `d.keys()`
- `d.values()`
- `d.items()`

# Practice question 1 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

print(a_book["Count of Monte Cristo"])
```

# Practice question 1 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

print(a_book["Count of Monte Cristo"])
```

Error

# Practice question 1 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

print(a_book["Count of Monte Cristo"])
```

Error

*Have to provide a key, not a value*

# Practice question 2 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

print("Count of Monte Cristo" in a_book)
```

# Practice question 2 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

print("Count of Monte Cristo" in a_book)
```

```
False
```

# Practice question 2 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

print("Count of Monte Cristo" in a_book)
```

False

*Looks at keys, not in values*

# Practice question 3 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

print(a_book["Name"])
```

# Practice question 3 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

print(a_book["Name"])
```

```
Count of Monte Cristo
```

*Provide a key, get the corresponding value back*

# Practice question 4 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

b_book = a_book
print(b_book["Pages"])
```

# Practice question 4 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

b_book = a_book
print(b_book["Pages"])
```

928

# Practice question 5 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

b_book = a_book
b_book["Name"] = "Count of UVA"
print(a_book["Name"] == "Count of Monte Cristo")
```

# Practice question 5 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

b_book = a_book
b_book["Name"] = "Count of UVA"
print(a_book["Name"] == "Count of Monte Cristo")
```

```
False
```

# Practice question 5 - *What is the output?*

```
a_book = {
  "Name": "Count of Monte Cristo",
  "Pages": 928,
  "Paperback?": True
}

b_book = a_book
b_book["Name"] = "Count of UVA"
print(a_book["Name"] == "Count of Monte Cristo")
```

False

*Make a copy of a_book. This is a reference to a_book. So when you change b_book you are also changing a_book*

# Practice Doing These Things with Dictionaries

- To get more comfortable with dictionaries, try the following:
  - Create a dictionary with existing key-value pairs
  - Create an empty dictionary and add to it
  - Load a dictionary through input statements
  - Try to access a key that is not in the dictionary
  - Check to see if a key exists before retrieving a value
  - Loop through a dictionary: by keys, by values, by key-value pairs

# Example of looping through dictionaries

```
# I'm going to pick up hamburgers for all my friends.
# I need a way to store all of the orders.
# Similar to lists and strings, dictionaries are collections.
# The things in the dictionary are going to be key-value pairs.
# For my orders dictionary, the items will be person-burger pairs.
orders = {'Sofiya':'cheese burger', 'Jacob':'bbq burger', 'Kat':'mushroom burger', 'Xinyu':'cheese burger'}
# one way to print out the burgers
for person in orders: # looping through the keys
    # print(person) # person is the key
    print(orders[person]) # print the value that is stored at person
for person in orders.keys(): # another way to loop through keys, no different than the loop above
    print(orders[person])
print('a second way to access values')
for burger in orders.values():
    print(burger)
print('a third way to access values')
for pair in orders.items(): # Each pair is a tuple - (person, value)
    print(pair[1]) # the second item in the tuple
# Let's make a new dict to store the prices/costs
costs = {} # keys are people, the values are costs
for person in orders:
    costs[person] = float(input("how much does "+ person + ", owe? "))
print(costs)
```

```python
d = {}  # an empty dictionary named d
print(type(d), d)

d = {4: "hi", 7: "bye"}  # A dictionary with 2 key-value pairs. The keys
# are integers, the values are strings.
print(type(d), d)

d = {3: "banana", "Scores": [3,6,7] }  # A dictionary with 2 key-value
# pairs. The keys are different types.
print(type(d), d)

d[12] = "apple"  # adding a new key-value pair to a dictionary
print(type(d), d)

x = d[12]  # retrieving a value from a dictionary. X will be assigned
“apple”
print(type(x), x)
print(type(d), d)  # notice nothing was removed

d[12] = "pumpkin"
print(type(d), d)  # the value of key 12 was changed, a new key-value
pair was not added
# retrieve a value from a dictionary. x will be assigned “apple”.
Nothing was removed from d
x = d[12]

# deleting a dictionary element
del d[12]

# How many key-value pairs in the dictionary
len(d)


# ---------------------------------------------------

# How to create an empty collection
my_list = []
my_tuple = ()
my_string = ""
my_dictionary = {}
```

```python
# Add elements to a dictionary
painting_years = {}
painting_years["Mona Lisa"] = 1503  # Leonardo da Vinci
painting_years["Girl With A Pearl Earring"] = 1665  # Johannes
painting_years["Starry Night"] = 1889  # Vincent van Gogh


# Print every painting with the year it was painted
for i in painting_years.keys():
    print(i, "was painted in the year", painting_years[i])

'''
# Age every celebrity one year
# Assume we have a dictionary called celebrity_ages that had
# the celebrity name as the key and their age as the value

for i in celebrity_ages.keys():
   celebrity_ages[i] += 1
'''


# numbers and their squares
number_squares = {}
for i in range(1,101):
   number_squares[i] = i ** 2

print(number_squares[13])  # should be 169
```

# Python Demonstration

Let's jump on PyCharm!

`dictionaries.py` - Examples illustrating the dictionary data structure.


# In-Class "lab" Activity!

# Activity for Today!

- In pairs or groups up to three work on the following activity.
- **dictionaries_ica1.py**
- *Write an English-to-Spanish translation program using a dictionary*

*Remember to check-in with a TA before leaving class today!*

# Reminder: CS Laptop Loaner Program

- This course requires students to have a **laptop**
- I realize that not everybody might have one (nor necessarily need one for their desired major / path…)
- If you do not have a laptop for any reason… ***not to worry!***
- The CS department's Systems staff has a notebook / laptop loaner program and will be able to loan you a notebook / laptop computer for the duration of the semester if you don't have one or if you cannot afford one.
  - Also available if your laptop is broken and under repair, we can arrange for you to receive a loaner laptop for a week or two until your own laptop is fixed

Interested? Link: https://www.cs.virginia.edu/wiki/doku.php?id=cs_laptop_loaner

*I am happy to be your sponsor. Please let me know.*