Intro to Computer Science
CS-UH 1001, Spring 2022

Lecture 10 –Dictionaries

Midterm Exam Schedule

- **Midterm Review:**
  March 7

- **Midterm Exam:**
  March 9

Recap

- Functions:
  - Defining functions:

```
def function_name(arguments):
    # function body
```

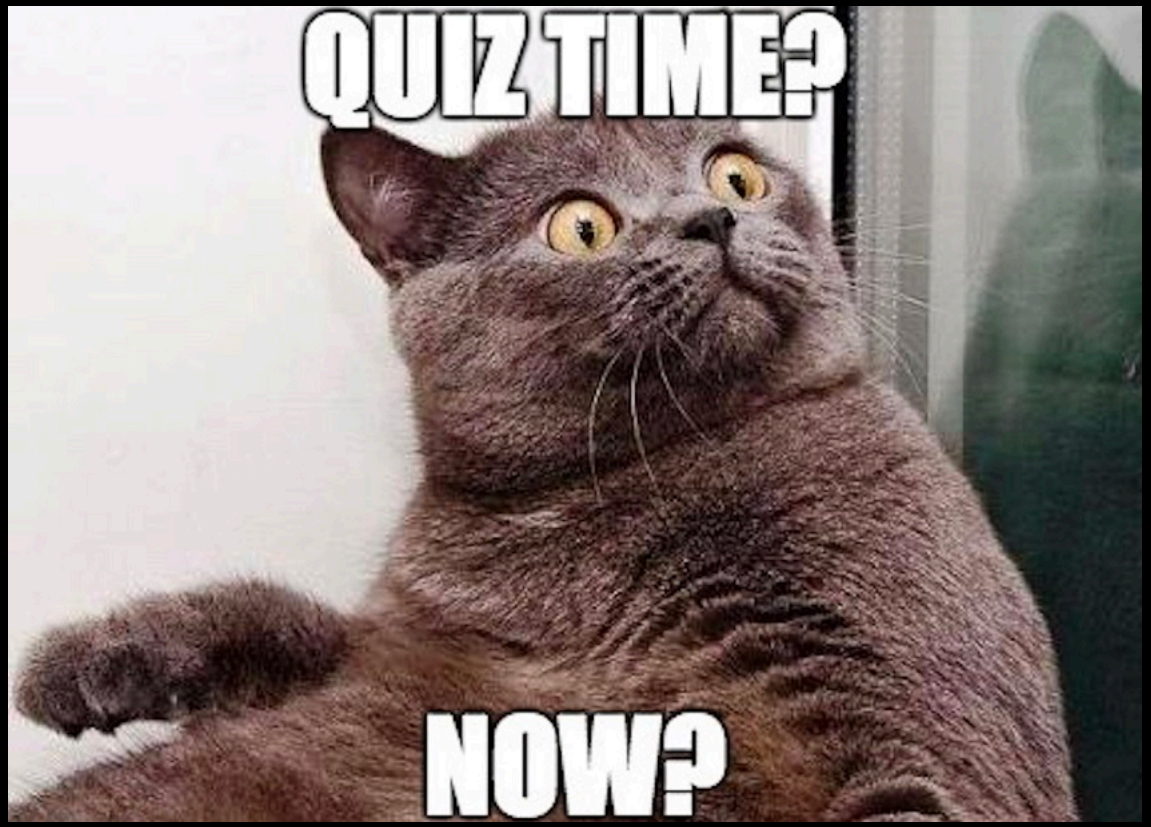
  - Defining functions with default arguments:

```
def function_name(arg=defaultvalue):
    # function body
```

- Two kinds of functions:
  - Void functions: do not return a value (None)
  - Fruitful functions: return a value
- Functions can have multiple return statements

Recap

- If an immutable variable is assigned a value anywhere within the function's body, it is a local variable
- A local function variable cannot be accessed outside the function they are defined in
- The global keyword allows to modify a variable outside of the current scope


QUIZ TIME?
NOW?

Modules

Modules

- We have seen that there are a number of helpful modules that can be used
  - random.randint() to generate random numbers
  - time.sleep(1) to pause the program
  - os.system("clear") to clear the screen
- Modules contains a set of functions that can be used
  - use them by importing their module using the *import* modulname

Modules

- The *import* statement should be added to the top of the program
  - import modulename

- Then you can use the functions of that module by using the modulename.function()

Example:

```
import random
num = random.randint(0,1)
```

math Module (import math)

- The *math* module has a number of useful math functions

- Some examples of the *math* functions are:
  - sqrt(x) returns the square root of a number
  - exp(x) returns the e^x
  - log(x) returns the natural logarithm of x
  - log10(x) returns the base-10 logarithm of x
  - sin(x) returns the sin of x

- A full list of the *math* module functions can be found here: https://docs.python.org/3/library/math.html

Functions and Modules

- A module is simply a file that contains a set of functions
- You can create your own module to break a large program into small manageable parts
- You can also reuse the modules in other programs and code, rather than typing them again
- Modules can also import other modules

Functions and Modules

- How to create a module:
  - Create a separate .py file
  - The name of the file will be the module name
  - In the module you can define the functions
  - Then import the module and use its functions

- Note: The module file has to be in the same folder as your Python file!

Example: *circle* module

- The circle module contains the following functions:
  - Calculate area of a circle
  - Calculate the diameter of a circle

- First create a file called circle.py
- Implement two functions: one for the area and one for the diameter

circle.py File

```
import math

def area(radius):
    ''' This function is used to calculate the area of a circle. It takes one argument
    called "radius" which can be an int or a float. It returns a float which is the
    area of the circle
    example: circle.area(5) '''

    return math.pi*radius**2

def diameter(radius):
    return 2*radius
```

Using the Module

```
import circle
```


```
radius = float(input(”Enter the radius of the circle: "))
print("The area of the circle is:", circle.area(radius))
print("The diameter of the circle is:", circle.diameter(radius))
```

Hands-on Session

Modules

Square Module (ex_10.1.py)

Create a module that has two functions to calculate the area and the diagonal of a square, where a is the length of a side

Formulas:

area = a*a

diagonal = a*math.sqrt(2)

Dictionaries

Dictionaries

- A dictionary is a collection of key-value pairs
- Dictionaries are defined using {}
- Dictionaries are mutable

- Each element in the dictionary consists of a key and a value (key-value pair)
  - grades = {key: value, key: value, key: value}

- The keys and values in an element can be any type
  - keys are unique and must be an immutable data type:
    - int, float, string
  - values can be immutable or mutable data types
    - int, float, string
    - list, dict

Creating a Dictionary

- Dictionaries are created using curly brackets {}

```
grades = {    keys      values
              'Jon':    9.5,
              'Robb':   10,
              'Arya':   10
          }
```

```
print(grades)      {'Jon': 9.5, 'Robb': 10, 'Arya': 10}
```

Creating a Dictionary

Dictionaries are mutable!

Dynamically creating a dictionary during runtime:

```
grades = {}    # create empty dictionary
grades['Jon'] = 9.5
grades['Robb'] = 10
grades['Arya'] = 10

print(grades)     {'Jon': 9.5, 'Robb': 10, 'Arya': 10}
```

Retrieving a Value

- Dictionaries are different from Lists:
  - You can not use numeric indices to access a value by its specific position
  - Instead, you need to use the key to retrieve a value

- dictionary_name[key] will retrieve the value associated with that key

```
value = dictionary_name[key]
```

Example: Lists and Dicts

```
student_list = ['Jon', 'Robb', 'Arya']
student_dictionary = {'Jon':10, 'Robb':5, 'Arya':9}

print(student_list)                 ['Jon', 'Robb', 'Arya']
print(student_dictionary)           {'Arya': 9, 'Robb': 5, 'Jon': 10}

print(student_list[0])              'Jon'
print(student_dictionary[0])        KeyError: 0
print(student_dictionary['Jon'])    10
print(student_dictionary['Cercei']) KeyError: 'Cercei'
```

How to Prevent Key Errors?

- Remember the in operator? It also works for dictionaries

```
if 'Cercei' in student_dictionary:
    print(student_dictionary['Cercei'])

if 'Cercei' not in student_dictionary:
    print("Cercei is not found")
```

Adding or Updating a key-value pair

- Adding a new key-value pair to a dictionary:
  - dictionary_name[key] = value
- Updating/overwriting a value in a dictionary:
  - dictionary_name[key] = new_value

Useful dictionary statements

- Deleting an key-value pair from a dictionary:
  - del dictionary_name[key]
- Number of key-value pairs in a dictionary:
  - len(dictionary_name)

Hands-on Session

Dictionary

Dictionary exercise (ex_10.1.py)

- Write a Python program that prints a dictionary where the keys are numbers between 1 and 15 and the values are the square of the keys.

- Expected output:
  print(my_dict)

  {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11: 121, 12: 144, 13: 169, 14: 196, 15: 225}

```
def take_a_break(minutes):
    time.sleep(minutes * 60)
    return

time_elapsed = 0   # time in minutes
class_ongoing = True
while class_ongoing == True:
    if time_elapsed == 75:
        take_a_break(15)

    time.sleep(60)
    time_elapsed = time_elapsed + 1
```

Iterating Through Dictionaries

- A for loop can be used to iterate through a dictionary :

```
dictionary = {'Jon':10, 'Robb':50, 'Arya':5}

for key in dictionary:
    print(key, dictionary[key])
```

```
Jon 10
Arya 5
Robb 50
```

- Note: Dictionaries are not designed to be in order!

Dictionary Methods

- .clear(): clears all key-value pairs in a dictionary
- .keys(): returns all keys of a dictionary
- .values(): returns all values of a dictionary
- .get(key, default_value): returns the value of a key; in case the value is not found it returns a default value
- .pop(key, default_value): removes the key-value pair from the dictionary and returns the value. If the key is not found, it returns the default value
- .popitem(): removes and returns the key-value pair that was last inserted

Dictionary Methods Examples

```
dictionary = {'Jon':10, 'Robb':50, 'Arya':5}

print (dictionary.get('Jon', 'not found'))
print (dictionary)

print (dictionary.keys())
print (dictionary.values())

print (dictionary.pop('Jon', 'not found'))
print (dictionary.pop('Jon', 'not found'))

key, value = dictionary.popitem()
print (key, value)

dictionary.clear()
print(dictionary)
```

10

{'Jon': 10, 'Robb': 50, 'Arya': 5}

dict_keys(['Jon', 'Robb', 'Arya'])

dict_values([10, 50, 5])

10

not found

Either: Robb 50
Or: Arya 5

{}

Hands-on Session

Character Count & Birthday Dictionary

Character frequencies (ex_10.2.py)

- Write a function that accepts a string as an argument and returns a dictionary where the keys are the characters of the string and the values are the character count

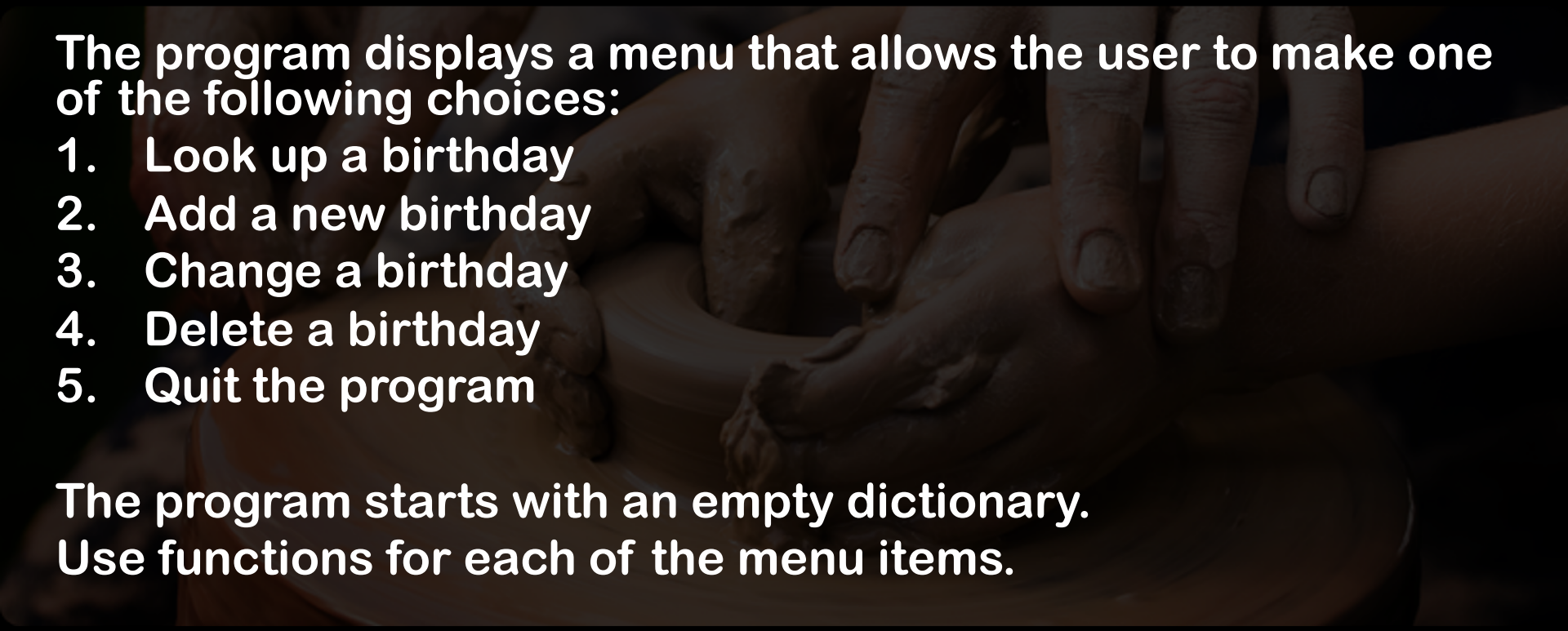
- Example:

  my_string = “Hello World”

  print(char_frequencies(my_string))
  {'H': 1, 'e': 1, 'l': 3, 'o': 2, ‘ ': 1, 'W': 1, 'r': 1, 'd': 1}

Birthday application (ex_10.3.py)

Write a program that keeps your friends' names and their birthdays stored in a dictionary. You will use the program to look up your friends' birthdays by entering their names.



The program displays a menu that allows the user to make one of the following choices:

1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

The program starts with an empty dictionary.
Use functions for each of the menu items.

Next Class

- We will look into:
  - File input and output
  - CSV files