Miscellaneous Topics

Contents

- Exception Handling
- Selecting Random Values
- Swapping
- Shortcuts
- Recursion

Exception Handling

- Exceptions occur due to circumstances beyond programmer's control
  - E.g., when invalid data are input or file cannot be accessed
- Even though the user is at fault
  - Programmer must anticipate exceptions
  - Include code to work around the occurrence

```
numDependents = int(input("Enter number of dependents: "))
taxCredit = 1000 * numDependents
print("Tax credit:", taxCredit)
```

[Run: When the user enters the word TWO]

```
Enter number of dependents:
ValueError: invalid literal for int() with base 10: ''
```

Exception Handling

- Some common exceptions:

Exception Name	Description and Example
AttributeError	An unavailable functionality (usually a method) is requested for an object. `(2, 3, 1).sort()` or `print(x.endswith(3))` `# where x = 23`
FileNotFoundError	Requested file doesn't exist or is not located where expected. `open("NonexistantFile.txt", 'r')`
ImportError ModuleNotFoundError	Import statement fails to find requested module. `import nonexistentModule`
IndexError	An index is out of range. `letter = "abcd"[7]`
KeyError	No such key in dictionary. `word = d['c'])` `# where d = {'a':"alpha", 'b':"bravo"}`

Exception Handling

- Some common exceptions:

KeyError	No such key in dictionary. `word = d['c'])  # where d = {'a':"alpha", 'b':"bravo"}`
NameError	The value of a variable cannot be found. `term = word  # where word was never created`
TypeError	Function or operator receives the wrong type of argument. `x = len(23)` or `x = 6 / '2'` or `x = 9 + 'W'` or `x = abs(-3,4)`
ValueError	Function or operator receives right type of argument, but inappropriate value. `x = int('a')` or `L.remove(item)  # where item not in list`
ZeroDivisionError	The second number in a division or modulus operation is 0. `num = 1 / 0` or `num = 23 % 0`

The *try* Statement

- The previous exception can be handled more robustly by protecting the code with a **`try`** statement

```
try:
    numDependents = int(input("Enter number of dependents: "))
except ValueError:
    print("\nYou did not respond with an integer value.")
    print("We will assume your answer is zero.")
    numDependents = 0
taxCredit = 1000 * numDependents
print("Tax credit:", taxCredit)
```

[Run: When the user enters the word TWO]

```
Enter number of dependents:

You did not respond with an integer value.
We will assume your answer is zero.
Tax credit: 0
```

The *try* Statement

- Three types of except clauses:

except:	(Its block is executed when any exception occurs.)
except *ExceptionType*:	(Its block is executed only when the specified type of exception occurs.)
except *ExceptionType* as *exp*:	(Its block is executed only when the specified type of exception occurs. Additional information about the problem is assigned to *exp*.)

The *try* Statement

- Program with different assumptions on exceptions:

```
def main():
    ## Display the reciprocal of a number in a file.
    try:
        fileName = input("Enter the name of a file: ")
        infile = open(fileName, 'r')
        num = float(infile.readline())
        print(1 / num)
    except FileNotFoundError as exc1:
        print(exc1)
    except ValueError as exc2:
        print(exc2)

main()
```

[Run: When `Numbers.txt` does not exit]

```
Enter the name of a file: Numbers.txt
[Errno 2] No such file or directory: 'Numbers.txt'
```

The *try* Statement

[Run: When the first line of **`Numbers.txt`** contains the word TWO]

```
Enter the name of a file: Numbers.txt
ValueError: could not convert string to float: 'TWO\n'
```

[Run: When the first line of **`Numbers.txt`** contains the number 2]

```
Enter the name of a file: Numbers.txt
0.5
```

The *try* Statement

- Program that uses exception handling to guarantee a proper response from the user:

```
def main():
    ## Request that the user enter a proper response.
    phoneticAlpha = {'a':"alpha", 'b':"bravo",'c':"charlie"}
    while True:
        try:
            letter = input("Enter a, b, or c: ")
            print(phoneticAlpha[letter])
            break
        except KeyError:
            print("Unacceptable letter was entered.")
main()
```

[Run]

```
Enter a, b, or c: d
Unacceptable letter was entered.
Enter a, b, or c: b
bravo
```

The *else* and *finally* Statement

- The following program uses exception handling to cope with the possibilities that the file is not found, the file contains a line that is not a number, or the file is empty

```
def main():
    ## Calculate the average and total of the numbers
    ## in a file.
    total = 0
    count = 0
    foundFlag = True
    try:
        infile = open("Numbers.txt", 'r')
    except FileNotFoundError:
        print("File not found.")
        foundFlag = False
```

The *else* and *finally* Statement

```
if foundFlag:
    try:
        for line in infile:
            count += 1
            total += float(line)
        print("average:", total / count)
    except ValueError:
        print("Line", count,
              "could not be converted to a float")
        if count > 1:
            print("Average so far:", total / (count - 1))
            print("Total so far:", total)
        else:
            print("No average can be calculated.")
    except ZeroDivisionError:
        print("File was empty.")
    else:
        print("Total:", total)
    finally:
        infile.close()
```

The *else* and *finally* Statement

- **`try`** statement can also include a single **`else`** clause
  - Follows the except clauses
  - Executed when no exceptions occur
- **`try`** statement can end with a **`finally`** clause
  - Usually used to clean up resources such as files that were left open
- **`try`** statement must contain either an **`except`** clause or a **`finally`** clause

Selecting Random Values

- The random module contains functions that randomly select items from a list and randomly reorder the items in a list

```
import random

elements = ["earth", "air", "fire", "water"]
print(random.choice(elements))
print(random.sample(elements, 2))
random.shuffle(elements)
print(elements)
print(random.randint(1, 5))  # random integer from 1 to 5
```

[Run]

```
earth
['air', 'earth']
['fire', 'water', 'air', 'earth']
5
```

Selecting Random Values

```
import random
import pickle

infile = open("DeckOfCardsList.dat", 'rb')
deckOfCards = pickle.load(infile)
infile.close()
print(deckOfCards)
print()
pokerHand = random.sample(deckOfCards, 5)
print(pokerHand)
```

[Run]

```
['2♠', '3♠', '4♠', '5♠', '6♠', '7♠', '8♠', '9♠', '10♠',
'J♠', 'K♠', 'Q♠', 'A♠', '2♥', '3♥', '4♥', '5♥', '6♥', '7♥',
'8♥', '9♥', '10♥', 'J♥', 'K♥', 'Q♥', 'A♥', '2♣', '3♣',
'4♣', '5♣', '6♣', '7♣', '8♣', '9♣', '10♣', 'J♣', 'K♣',
'Q♣', 'A♣', '2◆', '3◆', '4◆', '5◆', '6◆', '7◆', '8◆', '9◆',
'10◆', 'J◆', 'K◆', 'Q◆', 'A◆']

['8♣', 'K♥', '10◆', 'Q♣', 'K♠']
```

Selecting Random Values

```
## Items are selected from a list of six items.
## Selection probabilities are made different using
## if-elif-else statement.

import random

def main():
    for i in range(3):
        outcome = spinWheel()
        print(outcome, end="   ")

def spinWheel():
    n = random.randint(1, 20)
    if n > 15:
        return "Cherries"
    elif n > 10:
        return "Orange"
    elif n > 5:
        return "Plum"
```

Selecting Random Values

```
    elif n > 2:
        return "Melon"
    elif n > 1:
        return "Bell"
    else:
        return "Bar"

main()
```

[Run]

```
Melon  Cherries  Orange
```

- A float value can be chosen randomly from an interval by using

  `random.uniform(a, b)`

  where `a` and `b` are the lower bound and upper bound, respectively

Swapping

- In most programming languages, we need to introduce a third variable when swapping the values of two variables, **`x`** and **`y`**:

```
temp = x

x = y

y = temp
```

- Python, however, provides a nice shortcut:

```
x, y = y, x
```

  - Comma indicates that a tuple should be constructed
  - All the expressions to the right of the assignment operator are evaluated before any of the assignments are made

Shortcuts

- Assignment shortcuts:

```
a = 0
b = 0
c = 0
```

⇔

```
a = b = c = 0
```

  - When L is a list with three elements in it:

```
x = L[0]
y = L[1]
z = L[2]
```

⇔

```
x, y, z = L
```

  - We can assign three variables at a time:

```
x, y, z = 1, 2, 3
```

  - We can swap variables like below:

```
x, y, z = y, z, x
```

Shortcuts

- Shortcuts with condition:

```
if a == 0 and b == 0 and c == 0:
```
⇔
```
if a == b == c == 0:
```

```
if 1 < a and a < b and b < 5:
```
⇔
```
if 1 < a < b < 5:
```

Recursion

- A recursive solution to a problem has the general form:

If a base case is reached
    Solve the base case directly
else
    Repeatedly reduce the problem to a version increasingly close to a base case until it becomes a base case

```
def power(r, n):
    if n == 1:
        return r
    else:
        return r * power(r, n - 1)

print(power(2, 3))
```

[Run]

```
8
```

Recursion

- A word is a palindrome if it reads the same forward and backward (e.g., racecar, kayak, pullup)

```
def isPalindrome(word):
    # Convert all letters to lowercase.
    word = word.lower()
    # Words of zero or one letters are palindromes.
    if len(word) <= 1:
        return True
    elif word[0] == word[-1]:  # First and last letters match.
        # Remove first and last letters.
        word = word[1:-1]
        return isPalindrome(word)
    else:
        return False
```