Information, Computation, Communication

Learning Python

Examples, Exam-Like Questions

Agenda

- Types and Operators
- Dictionaries
  - 2a: Dictionaries and loops
  - 2b: Dictionaries and recursion
- Mutability, variable scope
- One-line code solutions

© kras99 / Adobe Stock

Types and Operators

© kras99 / Adobe Stock

Example 1: Types and Operators

- What is the type of the variable **`my_var`**?
- What does this program print?

```
a = -2000
b =  4001
c =  6002
my_var = c // a, b or a, c % a
c, b, a = my_var
print(a, b, c)
```

```
Q2: Suggested answers

(a)-4 True 2

(b)-1998 4001 -4

(c) 2 True -3

(d)-1998 4001 -3
```

Solution 1: Types and Operators

EXAMPLES

```
# my_var type is tuple
# Below is a shorter equivalent version without using my_var
a = -2000
b = 4001
c = 6002
c, b, a = c // a, b or a, c % a
# c // a => 6002 // (-2000)
#   = -4 (the result of integer division is rounded toward lower values!)
# b or a => 4001 or -2000
#   = 4001 (the first non zero number if True)
# c % a => 6002 % (-2000)
#   = 6002 – (-4)*(-2000) = -1998
# c, b, a = -4, 4001, -1998
print(a, b, c) # -1998 4001 -4
```

Dictionaries

© kras99 / Adobe Stock

Example 2a: Dictionaries

- What does this code do (explain the functionality)?
- What does it print?

```
s = "Abrakadabra"
d = {}
for c in s:
  if c in d.keys():
    d[c] = d[c] + 1
  else:
    d[c] = 1
print(d)
```

EXAMPLES

Solution 2a: Dictionaries

It counts unique characters in the given string and fills in a dictionary.

```
s = "Abrakadabra"
d = {}  # create an empty dictionary
for c in s:  # for every character in string s
  if c in d.keys():  # if char c is already a key in d
    d[c] = d[c] + 1  # increment the corresponding value
  else:  # if c is NOT already a key in d
    d[c] = 1  # create new key-value pair; initialize the value to 1
print(d)
# Answer: d keeps the total count of unique characters in s
# {'A': 1, 'b': 2, 'r': 2, 'a': 4, 'k': 1, 'd': 1}
# d is unordered; only key-value pairs matter to be correct, not the order
```

Recursion

© kras99 / Adobe Stock

Example 2b: Dictionaries and Recursion

Write a recursive function **`count_chars_recursive(s)`**, which takes a string, counts the unique characters in the string, and returns a corresponding dictionary

```
# Example usage
s = "Abrakadabra"
d_recursive = count_chars_recursive(s)
print(d_recursive)
```

Solution 2b: Dictionaries and Recursion

- In every **new** recursive call:
  - Read one new character
  - Update the dictionary accordingly
  - Make a recursive call with the part of the string not yet analyzed
- **Base case** (when not to make a recursive call?):
  - When the current function call is reading and analyzing the last character
- Will have to **create the dictionary** and pass it as the function argument to the recursive calls
  - Create an empty dictionary to start with
  - ***Dictionaries are mutable*** *(just like lists), so every function call will be able to modify our dictionary when passed as the function argument*

Example 2b: Dictionaries and Recursion

```python
# Example usage
s = "Abrakadabra"
d_recursive = count_chars_recursive(s)
print(d_recursive)
```

- Recall the usage example and notice that the first call to the recursive function takes only the string as the argument…
  - What about the dictionary that needs to be read and updated?
  - What about the index of the string to know which character to read?
    - *We will make them function arguments and assign them a default value to be used when nothing else is specified*

EXAMPLES

Solution 2b: Dictionaries and Recursion

```
def count_chars_recursive(s, index=0, d=None):
    if d is None:
        d = {}  # Create a dictionary before updating it
    # Base case: we've reached the end of the string
    if index == len(s):
        return d

    c = s[index]  # Current character
    if c in d:
        d[c] += 1
    else:
        d[c] = 1
    # Recursive call for the next character
    return count_chars_recursive(s, index + 1, d)
```

Solution 2b: Dictionaries and Recursion

```
def count_chars_recursive(s, index=0, d=None):
    if d is None:
        d = {}

    if index == len(s):
        return d

    c = s[index]
    if c in d:
        d[c] += 1
    else:
        d[c] = 1

    return count_chars_recursive(s, index + 1, d)
```

1st call: `count_chars_recursive("Abrakadabra")`
- index = 0, d = None → d = { }, c = 'A', d['A'] = 1, d = {'A':1}

2nd call: `count_chars_recursive("Abrakadabra", 1, d)`
- index = 1, c = 'b', d['b']=1, d = {'A':1, 'b':1}

3rd call: `count_chars_recursive("Abrakadabra", 2, d)`
- index = 2, c = 'r', d['r'] = 1, d = {'A':1, 'b':1, 'r':1}

4th call: `count_chars_recursive("Abrakadabra", 3, d)`
- index = 3, c = 'a', **d['a'] = 1**, d = {'A':1, 'b':1, 'r':1, 'a':1}

5th call: `count_chars_recursive("Abrakadabra", 4, d)`
- index = 4, c = 'k', d['k'] = 1, d = {'A':1, 'b':1, 'r':1, 'a':1, 'k':1}

6th call: `count_chars_recursive("Abrakadabra", 5, d)`
- index = 5, c = 'a', **d['a'] = 2**, d = {'A':1, 'b':1, 'r':1, 'a':2, 'k':1}

…

12th call: `count_chars_recursive("Abrakadabra", 11, d)`
- return d

Mutability, Variable Scope

© kras99 / Adobe Stock

Example 3: Mutable Objects, Variable Scope

```python
# Let's manage a candy stash
candies = 10


# Function definitions
```

```python
def restock_candies(amount):
    candies = amount
    return f"Restocked to {candies} candies."
```

```python
def eat_candies(amount):
    global candies
    if candies >= amount:
        candies -= amount
        return f"Ate {amount} candies.
                  Remaining stash: {candies}."
    else:
        return f"Not enough candies to eat {amount}!
                  Stash: {candies}."
```

```python
def share_candies(bag):
    bag.append("shared")
    return f"Candies in the bag after sharing: {bag}"
```

```python
print(restock_candies(50))

print(eat_candies(3))
bag_of_candies = ["chocolate", "gum"]

print(share_candies(bag_of_candies))

print(f"Bag of candies after sharing: {bag_of_candies}")
```

Solution 3: Mutable Objects, Variable Scope

```
# Let's manage a candy stash!
candies = 10


# Function definitions





print(restock_candies(50))
```

```
def restock_candies(amount):
    candies = amount
    return f"Restocked to {candies} candies."
```

Code execution:

- local variable amount = 50
- local variable candies = amount = 50
- `Restocked to 50 candies`

Solution 3: Mutable Objects, Variable Scope

```
# Let's manage a candy stash!
candies = 10


# Function definitions
```

```
def eat_candies(amount):
    global candies
    if candies >= amount:
        candies -= amount
        return f"Ate {amount} candies.
                 Remaining stash: {candies}."
    else:
        return f"Not enough candies to eat {amount}!
                 Stash: {candies}."
```

```
print(restock_candies(50))
print(eat_candies(3))
```

Code execution:

- local variable amount = 3
- **global** variable candies = 10
- `if` condition evaluates to True
    - candies = candies − 3 = 7
    - `Ate 3 candies. Remaining stash: 7.`

Solution 3: Mutable Objects, Variable Scope

```
# Let's manage a candy stash!
candies = 10

# Function definitions
```

```
def share_candies(bag):
    bag.append("shared")
    return f"Candies in the bag after sharing: {bag}"
```

Code execution:

- local variable bag = ["chocolate", "gum"]
- bag = ["chocolate", "gum", "shared"]
- `Candies in the bag after sharing: ["chocolate", "gum", "shared"]`

```
print(restock_candies(50))
print(eat_candies(3))
bag_of_candies = ["chocolate", "gum"]
print(share_candies(bag_of_candies))
print(f"Bag of candies after sharing: {bag_of_candies}")
```

Solution 3: Mutable Objects, Variable Scope

```
# Let's manage a candy stash!
candies = 10
# Function definitions
# …
# …
bag_of_candies = ["chocolate", "gum"]
print(share_candies(bag_of_candies)) # bag_of_candies modified by the function
print(f"Bag of candies after sharing: {bag_of_candies}")
# …
# …
# Candies in the bag after sharing: ['chocolate', 'gum', 'shared']
# Bag of candies after sharing: ['chocolate', 'gum', 'shared']
```

One-Line Solutions

© kras99 / Adobe Stock

Example 4: Removing Duplicates & Sorting

Write **one** line of Python code that transforms a string `s` into a list `sorted_chars` containing **unique** characters from `s` (no repetitions) sorted in **reverse** alphabetical order.

EXAMPLES

Solution 4: Removing Duplicates & Sorting

```python
# Answer (complete script)
# Input string
s = "crepes are awesome"

# Convert string to a set to remove duplicates
# Convert set to a list and sort it in reverse alphabetical order
sorted_chars = sorted(list(set(s)), reverse=True)

# Print the result
print(sorted_chars)
# ['w', 's', 'r', 'p', 'o', 'm', 'e', 'c', 'a', ' ']
```

Next:
Final Exam

© kras99 / Adobe Stock