Currency Project

Lucy Harlow

08/03/2021

My first Python project takes as its input some amount of money, and returns that amount sorted into coins of a given currency. I began this project a little over a month ago, as a total beginner, after first learning the most basic Python arithmetic operations. This document begins with the code in its current form, then summarises my progress up to this point.

Current program

I wrote this as a function that takes as inputs some amount of money (as an integer, in the smallest denomination), and a dictionary of the currency the user wishes to test. By separating the keys and values into lists, creating an empty list (numcoin), then appending to that empty list the values generated by the for loop, I was able to generate an output in the form of a new dictionary:

```python
def makeChange(cash, currency):
    '''
    INPUT
    Takes two variables, an integer and a dictionary.
    cash is the amount of money in units of the smallest available unit, e.g. pennies.
    currency has coin names as keys and values as their associated values, e.g. '£2': 200.

    OUTPUT
    Returns another dictionary: keys are coin names and values are numbers of those coins
    in the cash variable.
    '''
    denom = list(currency.values())
    names = list(currency.keys())
    numcoin = []

    for amount in denom:
        if cash >= amount:
            numcoin.append(cash // amount)
            cash = cash % amount
        else:
            numcoin.append("None")
    result = dict(zip(names, numcoin))
    return result


'''
Dictionaries of some currency variables:

currency_uk = {'£2': 200, '£1': 100, '50p': 50, '20p': 20, '10p': 10, '5p': 5, '2p': 2,
'1p': 1}
```

```
currency_us = {'Quarters': 25, 'Dimes': 10, 'Nickels': 5, 'Pennies': 1}

currency_euro = {'€2': 200, '€1': 100, '50c': 50, '20c': 20, '10c': 10, '5c': 5, '2c': 2,
'1c': 1}
'''

#cash = int(input("Cash in smallest currency unit: "))

#print(makeChange(cash, currency_uk))
```

Initially, I was concerned with the formatting of the output, and included in the body of the program various cumbersome ways of displaying the results. Today I wrote the following function, which displays a dictionary more readably, with key/value pairs on separate lines:

```
def dictionaryDisplay(myDict):
    '''
    Displays key/value pairs on separate lines
    '''
    keys = list(myDict.keys())
    values = list(myDict.values())
    dictzip = zip(keys, values)
    for key, value in zip(keys, values):
        print("{}: {}".format(key, value))

#coindisplay = makeChange(cash, currency_uk)

#dictionaryDisplay(coindisplay)
```

Past versions

Version 1

The logic of the problem hasn't changed since I first started this project, but I've learned a lot since then. I wrote the following version shortly after my first Python 'hello world', knowing nothing more than variable assignments, print, and simple arithmetic operations. The beginner breakthrough here, and the reason I wanted to include the very early experiments in this document, was the realisation that variables can be changed as you go along, enabling the formula itself to remain unchanged.

Even as I wrote these 33 lines I knew there was a better way – I was telling Python to do exactly the same thing over and over again, until it ran out of denominations, and I was doing it by retyping that unchanged formula every time I wanted to move on. I needed to know how to loop, but I hadn't learned that yet! This need was emphasised when I made a typo in one of the lines, and took a very long time to find it. My code was inefficient and simultaneously very susceptible to mistakes.

As you can see, I hadn't yet worked out a sensible way to display the results, and instead printed the variables after each operation. However, this taught me an extremely valuable lesson: inserting print statements is an excellent way of understanding what a program is doing, especially as the code becomes more complex and sophisticated.

```
cash = 573
print(cash)
GBP2 = (cash - (cash % 200)) / 200
print(GBP2)
cash = cash % 200
print(cash)
GBP1 = (cash - (cash % 100)) / 100
```

```
print(GBP1)
cash = cash % 100
print(cash)
GBP50 = (cash - (cash % 50)) / 50
print(GBP50)
cash = cash % 50
print(cash)
GBP20 = (cash - (cash % 20)) / 20
print(GBP20)
cash = cash % 20
print(cash)
GBP10 = (cash - (cash % 10)) / 10
print(GBP10)
cash = cash % 20
print(cash)
GBP05 = (cash - (cash % 5)) / 5
print(GBP05)
cash = cash % 5
print(cash)
GBP02 = (cash - (cash % 2)) / 2
print(GBP02)
cash = cash % 2
print(cash)
GBP01 = (cash - (cash % 1)) / 1
print(GBP01)
cash = cash % 1
print(cash)
```

Version 2

There are slight improvements here: I learned how to request a user input and convert it to integers, and I displayed the results all together by concatenating the denominations in the print statement. I also worked out that int() rounds any float down, which meant I no longer had to subtract the modulo from the original value in order to get the number I needed.

```
cash = int(input("Cash in pennies: "))
GBP2 = int(cash/200)
cash = cash % 200
GBP1 = int(cash/100)
cash = cash % 100
GBP50 = int(cash/50)
cash = cash % 50
GBP20 = int(cash/20)
cash = cash % 20
GBP10 = int(cash/10)
cash = cash % 10
GBP05 = int(cash/5)
cash = cash % 5
GBP02 = int(cash/2)
cash = cash % 2
GBP01 = int(cash/1)
cash = cash % 1
print(GBP2, GBP1, GBP50, GBP20, GBP10, GBP05, GBP02, GBP01)
```

Version 3

This version is a very slight improvement, using floor division instead of rounding down the division of 'cash' by the denomination, reducing two steps to one.

```python
user = int(input("Cash in pennies: "))
cash = user
GBP2 = cash // 200
cash = cash % 200
GBP1 = cash // 100
cash = cash % 100
GBP50 = cash // 50
cash = cash % 50
GBP20 = cash // 20
cash = cash % 20
GBP10 = cash // 10
cash = cash % 10
GBP05 = cash // 5
cash = cash % 5
GBP02 = cash // 2
cash = cash % 2
GBP01 = cash // 1
cash = cash % 1

print(GBP2, GBP1, GBP50, GBP20, GBP10, GBP05, GBP02, GBP01)
```

Version 4

Finally, I learned how to use loops in Python! However, I didn't understand them yet. Although this version works, the counter is redundant, and the variable 'coin' in the first line of the loop is never used. I was thinking in terms of while loops. Furthermore, I set up the formatting of the results within the loop, which made it difficult to make edits, not to mention easy to introduce errors while making those edits.

```python
cash = int(input("Cash in pennies: "))
denom = [200, 100, 50, 20, 10, 5, 2, 1]
result = ['£2', '£1', '50p', '20p', '10p', '5p', '2p', '1p']
count = 0


for coin in denom:
    if cash >= denom[count]:
        result[count] = (result[count] + " coins: " + str(cash // denom[count]))
        cash = cash % denom[count]
        count += 1
    else:
        result[count] = (result[count] + " coins: None")
        count += 1

print(result)
```

Version 5

This version improves upon the design of the loop in the previous example.

After writing this version, I moved on to the function described at the top of this document.

```python
cash = int(input("Cash in pennies: "))
denom = [200, 100, 50, 20, 10, 5, 2, 1]
coins = ["£2", "£1", "50p", "20p", "10p", "5p", "2p", "1p"]
numcoin = []

for value in denom:
    if cash >= value:
        numcoin.append(cash // value)
        cash = cash % value
    else:
        numcoin.append("None")

for coin, num in zip(coins, numcoin):
    print("{}: {}".format(coin, num))
```