py101-py109_sp_easy_1

December 4, 2024

1 PY 101 - PY 109

1.1 Small Problems

1.1.1 Easy 1

1.2 Isn't it Odd?

Write a function that takes one integer argument and returns `True` when the number's absolute value is odd, `False` otherwise.

```
[2]: def is_odd(num):
         return True if abs(num) % 2 != 0 else False
```

1.2.1 Isn't it Odd? Recommended Solution

This is a little cleaner and shorter than my solution.

```
def is_odd(number):
    return abs(number) % 2 == 1
```

1.3 Odd Numbers

Print all odd numbers from `1` to `99`, inclusive, with each number on a separate line.

Bonus question: Can you solve the problem by iterating over just the odd numbers?

P: The problem seems pretty straightforward. Print (don't return) odd numbers (numbers not divisible by two) between and including 1 and 99, with each number in a new, separate line. The bonus queston asks to solve the problem by just iterating over the odd numbers, which prevents brute forcing a loop and doing modulus division on everything.

E: No examples or test cases are provided

D: I'm not sure that identifying a data structure is relevant. I could produce a list and print from the list, but this is a short, simple function and I don't think that's necessary. A range is a data structure.

A: Ok, the simplest way to solve this is with modulus division by two, printing out everything that doesn't have a result of 0 on a new line, but the bonus questin throws that out. The way to iterate over only the odd numbers would be a slice, starting at 1 and finishing at 100 with a step of 2.

I don't want to type all the numbers in and I can't iterate through the odd numbers, so a range seems like the right way to do this.

Initialize a range, slice the range in a loop, print the sliced range one by one

[2]:
```
my_range = range(100)
for num in my_range[1:100:2]:
    print(num)
```

```
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
51
53
55
57
59
61
63
65
67
69
71
73
75
77
79
```

```
81
83
85
87
89
91
93
95
97
99
```

1.3.1 Odd Numbers Recommended Solution

I didn't actually know that you could add the step option to range() in a loop like that. I'll remember that.

```
for number in range(1, 100, 2):
    print(number)
```

1.4 Even Numbers

Print all even numbers from `1` to `99`, inclusive, with each number on a separate line.

Bonus question: Can you solve the problem by iterating over just the even numbers?

P: This seems pretty similar to the last problem. I'd say refer to those notes.

E: No examples or test cases are provided

D: I'm not sure that identifying a data structure is relevant. I could produce a list and print from the list, but this is a short, simple function and I don't think that's necessary. A range is a data structure.

A: Ok, the simplest way to solve this is with modulus division by two, printing out everything that has a result of 0 on a new line, but the bonus question throws that out. The way to iterate over only the odd numbers would be a slice, starting at 2 and finishing at 100 with a step of 2. I don't want to type all the numbers in and I can't iterate through the odd numbers, so a range seems like the right way to do this.

Initialize a range, slice the range in a loop, print the sliced range one by one

[3]:
```
for num in range(2,100,2):
    print(num)
```

```
2
4
6
8
10
12
14
```

```
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66
68
70
72
74
76
78
80
82
84
86
88
90
92
94
96
98
```

1.4.1 Even Numbers Recommended Solution

My solution is the same as the recommended solution.

1.5 How Big is the Room?

Build a program that asks the user to enter the length and width of a room, in meters, then prints the room's area in both square meters and square feet.

Note: 1 square meter == 10.7639 square feet

P: Ok, the problem wants user input of two variables, length and width in meters. It prints (not returns) the room's areas (lw) in square meters and square feet. We'll have to get both variables, get the area in square meters, then get the converted area, then print.

E: There are no test cases

D: I think I'm just printing strings here

A: Input for two variables. [skip data validation because of the scope.] create and assign area_in_metric = lw. *Create and assign area_in_imperial = area_in_metric* 10.7639. Print each in a single f-string.

[3]:
```python
length = float(input('Enter the length of the room in meters: '))
width = float(input('Enter the width of the room in meters: '))
print(str(length * width) + " in square meters, or " + str((length * width) *␣
  ↪10.7639) + " in square feet")
```

```
Enter the length of the room in meters:  10
Enter the width of the room in meters:  15

150.0 in square meters, or 1614.585 in square feet
```

1.5.1 Further Exploration

Modify the program to let the user specify the measurement type (meters or feet). Compute the area accordingly and print it and its conversion in parentheses.

[ ]:
```python
#l_temp = input('Enter the length of the room and \'m\' for meters, \'f\' for␣
  ↪feet: ')
length, l_measurement = (input('Enter the length of the room and \'m\' for␣
  ↪meters, \'f\' for feet. No comma, only two characters: ')).split()
width, w_measurement = (input('Enter the width of the room and \'m\' for␣
  ↪meters, \'f\' for feet. No comma, only two characters: ')).split()

length = float(length)
width = float(width)

if l_measurement == "f":
    length /= 3.28084
if w_measurement == "f":
    length /= 3.28084
```

```
print(str(length * width) + " in square meters, or " + str((length * width) *␣
  ↪10.7639) + " in square feet")
```

1.6 Tip Calculator

Create a simple tip calculator. The program should prompt for a bill amount and a tip rate. The program must compute the tip, then print both the tip and the total amount of the bill. You can ignore input validation and assume that the user will enter valid numbers.

P: This is a tip calculator - input for bill amount and tip rate, so two user-submitted variables. Compute the tip, add the tip to the bill, print both. Ignore data validation.

E: The example shows the questions and answers, then the tip and total.

D: No data structure is really relevant here. I'll be using floats down to the hundredths.

A: Input a, input b, a*b = tip, a+tip = total, print 'the tip is [tip]' and 'the total is [total]

C:

[ ]:
```
# Note - I read about the string formatting (:.2f) after a google search. I␣
  ↪need to learn/practice it more
bill = float(input('What is the bill? '))
tip_per = float(input('What is the tip percentage? '))

tip = bill * (tip_per / 100)
total = bill + tip

print(f'The tip is ${tip:.2f}')
print(f'The total is ${total:.2f}')
```

1.7 Sum or Product of Consecutive Integers

Write a program that asks the user to enter an integer greater than `0`, then asks whether the user wants to determine the sum or the product of all numbers between `1` and the entered integer, inclusive.

P: The problem asks for a few decision points (write a number, determine sum/product), then will loop through numbers in the range, adding or multiplying them as it goes along.

E: While it doesn't specify, the sample output shows the result being printed, not returned. It also shows "s" and "p" instead of typing out

D: I feel good about a single integer variable to hold the incrementing final value, a range, and a for loop.

A: Get input. for x in range (1, input), if s, final_value += x, else final_value *= x

[ ]:
```
user_num = int(input('Please enter an integer greater than 0: '))
sum_or_product = input('Enter "s" to compute the sum, or "p" to compute the␣
  ↪product: ')
```

```
final_num = 1

for x in range(2, (user_num+1)):
    if sum_or_product == "s" or sum_or_product == "sum":
        sum_or_product = "sum"
        final_num += x
    else:
        sum_or_product = "product"
        final_num *= x

print(f'The {sum_or_product} of the integers between 1 and {user_num} is␣
 ↪{final_num}.')
```

The solution offered set up a few functions. For the sum it directly ran through the range as in `sum(range(1, target_num+1))` which s nicer than mine. It also suggested building in some data validation, so I'll redo with that.

```
[ ]: def get_sum(user_num):
         return sum(range(1, user_num+1))

     def get_product(user_num):
         final = 1
         for x in range(1, user_num+1):
             final *= x
         return final

     def get_data():
         sum_or_product = input("Would you like to calculate the sum or product? (s/
      ↪p) ")
         while sum_or_product not in ["s", "p"]:
             print("Invalid input. Please enter 's' for sum or 'p' for product.")
             sum_or_product = input("Would you like to calculate the sum or product?␣
      ↪(s/p) ")

         user_num = None
         while user_num is None:
             user_num = input("Enter a number: ")
             if not user_num.isdigit() or int(user_num) <= 1:
                 print("Invalid input. Please enter an integer greater than zero")
                 user_num = None
                 continue
             else:
                 user_num = int(user_num)

         return user_num, sum_or_product

     user_num_out, sum_or_product_out = get_data()
```

```
if sum_or_product_out == "s":
    print("The sum of the integers between 1 and "
          f"{user_num_out} is {get_sum(user_num_out)}.")
else:
    print("The product of the integers between 1 and "
      f"{user_num_out} is {get_product(user_num_out)}.")
```

1.8 Short Long Short

Write a function that takes two strings as arguments, determins the length of the two strings, and then returns the result of concatenating the shorter string, the longer string, and the shorter string once again. You may assume that the strings are of different lengths.

P: Straightforward - input two strings. Count the length of the strings - print the concatenation of short+long+short. For validation purposes, assume the strings are actually going to be different.

E: The examples make clear what to do, and an empty string still counts

D: We just dealing with strings here

A: Input two strings as arguments in a function. Create a reference value of negative infinity. Compare len(str1) with len(str2) and set one as the biggun. return (short + long + short)

[ ]:
```
def short_long_short(str_1, str_2):
    if len(str_1) > len(str_2):
        long_string = str_1
        short_string = str_2
    else:
        long_string = str_2
        short_string = str_1
    return (short_string + long_string + short_string)

print(short_long_short('abc', 'defgh'))
print(short_long_short('abcde', 'fgh'))
print(short_long_short('', 'xyz'))
```

Man, the official solution is smooth, haha:

```
def short_long_short(string1, string2):
    if len(string1) > len(string2):
        return string2 + string1 + string2
    else:
        return string1 + string2 + string1
```

I also really like this person's answer:

```
def short_long_short(str1, str2):
    sorted_strs = sorted([str1, str2], key=len)
    return sorted_strs[0] + sorted_strs[1] + sorted_strs[0]
```

This one is also good:

```
def short_long_short(str1, str2):
    return str2 + str1 + str2 if len(str1) > len(str2) else str1 + str2 + str1
```

1.9 Leap Years (Part 1)

Write a function that takes any year greater than `0` as input and returns True if the year is a leap year or `False` if it's not. For simplicity, this exercise assumes that the Gregorian calendar has been in continuous use since the year 1. We'll address that assumption in the next exercise that follows this one.

To determine whether a given year is a leap year, use the rules of the Gregorian calendar:

- If the year is divisible by 400, it **is** a leap year
- If the year is divisible by 100 but not by 400, it is **not** a leap year
- If the year is divisible by 4 but not by 100, it **is** a leap year
- All other years **are not** leap years.

P: The problem seems pretty straightforward. Take a year as input and return true if it's a leap year, false if it's not. The examples show that it actually means return, not print.

E: The sample code shows the year as an argument passed into the function with a boolean value returned.

D: The return value is boolean.

A: if year % 400 == 0 return true, if year % 100 == 0 and year % 400 != 0 return false. If year % 4 == 0 and year % 100 != 0 return True. else return False

[ ]:
```
def is_leap_year(year):
    if year % 400 == 0:
        return True
    elif year % 100 == 0 and year % 400 != 0:
        return False
    elif year % 4 == 0 and year % 100 != 0:
        return True
    else:
        return False

# These examples should all print True
print(is_leap_year(1) == False)
print(is_leap_year(2) == False)
print(is_leap_year(3) == False)
print(is_leap_year(4) == True)
print(is_leap_year(1000) == False)
print(is_leap_year(1100) == False)
print(is_leap_year(1200) == True)
print(is_leap_year(1300) == False)
print(is_leap_year(1751) == False)
print(is_leap_year(1752) == True)
print(is_leap_year(1753) == False)
print(is_leap_year(1800) == False)
print(is_leap_year(1900) == False)
print(is_leap_year(2000) == True)
print(is_leap_year(2023) == False)
print(is_leap_year(2024) == True)
print(is_leap_year(2025) == False)
```

1.10 Leap Years Part 2

In the previous exercise, we assumed that the Gregorian calendar has been in conintuous use since the year 1. However, the Gregorian calendar wasn't adopted until much later. Prior to that, much of the world used the Julian calendar, which observed leap year every 4 years.

In 1752, England, Ireland, and the British colonies all switched to the Gregorian calendar. Update the function from the previous exercise so it uses the Julian calendar prior to 1752 and the Gregorian calendar starting in 1752.

[ ]:
```
def is_leap_year(year):
    if year < 1752:
        return year % 4 == 0
    if year >= 1752:
        if year % 400 == 0:
            return True
        if year % 100 == 0:
            return False
    return year % 4 == 0
# These examples should all print True
print(is_leap_year(1) == False)
print(is_leap_year(2) == False)
print(is_leap_year(3) == False)
print(is_leap_year(4) == True)
print(is_leap_year(1000) == True)
print(is_leap_year(1100) == True)
print(is_leap_year(1200) == True)
print(is_leap_year(1300) == True)
print(is_leap_year(1751) == False)
print(is_leap_year(1752) == True)
print(is_leap_year(1753) == False)
print(is_leap_year(1800) == False)
print(is_leap_year(1900) == False)
print(is_leap_year(2000) == True)
print(is_leap_year(2023) == False)
print(is_leap_year(2024) == True)
print(is_leap_year(2025) == False)
```

1.11 Multiples of 3 and 5

Write a function that computes the sum of all numbers between `1` and some other number, inclusive, that are multiples of `3` or `5`. For instance, if the supplied number is `20`, the result should be `98 (3`

`+ 5 + 6 + 9 + 10 + 12 + 15 + 18 + 20)`

You may assume that the number passed in is an integer greater than 1

P: SImple enough - sum the numbers in a range that are multiples of 3 or 5. Return the number.

E: The example shows that the final value is returned

D: Take in integers, spit out integer. Use a range in the middle

A: multisum(num) = sum(range(1, num+1, 3)) + sum(range(1, num+1, 5))

[ ]:
```python
def of_3_or_5(number):
    return number % 3 == 0 or number % 5 == 0

def multisum(num):
    count = 0
    for x in range(1, (num+1)):
        if of_3_or_5(x):
            count += x
    return count

# These examples should all print True
print(multisum(3) == 3)
print(multisum(5) == 8)
print(multisum(10) == 33)
print(multisum(1000) == 234168)
```

1.12 UTF-16 String Value

Write a function that determines and returns the UTF-16 string value of a string passed in as an argument. The UTF-16 string value is the sum of the UTF-16 values of every character in the string. (You may use `ord` to determine the UTF-16 value of a character.)

P: The problem is straightforward. Write a function that 1. calculates the sum of all the UTF-16 values in the string, then 2. returns that value E: The examples show that this is a return, not a print. They also show how to work with non ASCII characters by setting a variable D: I'll be returning an integer A: Create a sum variable, loop through the string, adding the value of each character to the sum variable, return the sum variable

[ ]:
```python
def utf16_value(strang):
    ret_value = 0
    for char in strang:
        ret_value += ord(char)
    return ret_value

# These examples should all print True
print(utf16_value('Four score') == 984)
print(utf16_value('Launch School') == 1251)
print(utf16_value('a') == 97)
print(utf16_value('') == 0)
```

```
# The next three lines demonstrate that the code
# works with non-ASCII characters from the UTF-16
# character set.
OMEGA = "\u03A9"              # UTF-16 character 'Ω' (omega)
print(utf16_value(OMEGA) == 937)
print(utf16_value(OMEGA + OMEGA + OMEGA) == 2811)
```

2 PY 101 - PY 109

2.1 Small Problems

2.1.1 Easy 2

2.2 Welcome Stranger

Create a function that takes 2 arguments, a list and a dictionary. The list will contain 2 or more elements that, when joined with spaces, will produce a person's name. The dictionary will contain two keys, `"title"` and `"occupation"`, and the appropriate values. Your function should return a greeting that uses the person's full name, and mentions the person's title.

```
greeting = greetings(
    ["John", "Q", "Doe"],
    {"title": "Master", "occupation": "Plumber"},
)
print(greeting)
# Hello, John Q Doe! Nice to have a Master Plumber around.
```

P: Create a function that takes a related list and dictionary. The list has 'at least two' elements that make a full name when joined with spaces. The dictionary has that person's title and occupation. {title: x, occupation: y}

E: The example shows the function taking both arguments and returning them, usable in an f string

D: I'll be creating a string

A: `START  SET list, dictionary FOR items in list     if list.index('item') = -1          name_maker.append(item)     else          name_maker.append(item + " ") SET return_string f"Hello {name_maker}! Nice to have a {dictionary.get("title")} {dictionary.get("occupation")} around.`

```
[ ]: def greetings(lst, dct):
         name_string = " ".join(lst)
         return (f'Hello {name_string}! Nice to have a {dct.get("title")} {dct.
       ↪get("occupation")} around.')

     greeting = greetings(
         ["John", "Q", "Doe"],
         {"title": "Master", "occupation": "Plumber"},
     )
```

```
print(greeting)
```

The suggested solution put everything on the return line, which is simple.

status is interesting. I don't see it in the documentation under dictionaries.

2.3 Greeting a user

Write a program that asks for a user's name, then greets the user. If the user appends a ! to their name, the computer will yell the greeting (print it using all uppercase).

P. The problem is well-defined- take a name, print a greeting. If the name has !, print the greetiung in all caps

E: The examples show a prompt asking for a name, the name, and a greeting.

D: This is just strings

A: Prompt for name

Does name have '!"?

If no: nice quiet hello

If yes: LOUD HELLO

```
[ ]: name = input('What is your name? ')
     if name[-1] != '!':
         print(f'Hello {name}.')
     else:
         print('HELLO ' + name.upper())
```

2.4 Multiplying Two Numbers

Create a function that takes to arguments, multiplies them together, and returns the result.

P. Simple enough to grasp - two numbers, number * number, return

E. Exaple shows exactly what to expect

D. JUst using integers

A: RUn function with two parameters, assign product of parameters to variable, return variable

```
[ ]: def multiply(x,y):
         return x * y

     multiply(2,4)
```

2.5 Squaring an argument

Using the multiply function from the "Multiplying Two Numbers" exercise, write a function that computes the square of its argument (the square is the result of multiplying a number by itself).

[ ]:
```
def multiply(x,y):
    return x * y

def square(a):
    return multiply(a,a)

print(square(5))
```

[ ]:
```
def multiply(x,y):
    return x * y

def power(a, n):
    '''n=exponent'''
    return_value = 1
    for i in range(n):
        return_value = multiply(return_value, a)
    return return_value
print(power(2,3))
```

2.6 Floating Point Arithmetic

Write a program that prompts the user for two positive numbers (floating-point), then prints the results of the following operations on those two numbers: addition, subtraction, product, quotient, floor quotient, remainder, and power. Do not worry about validating the input.

[ ]:
```
float_1 = float(input('Write a floating point number: '))
float_2 = float(input('Write a floating point number: '))

return_dict = {}
return_dict["add"] = float_1 + float_2
return_dict["sub"] = float_1 - float_2
return_dict["multiply"] = float_1 * float_2
return_dict["divide"] = float_1 / float_2
return_dict["floor_div"] = float_1 // float_2
return_dict["mod"] = float_1 % float_2
return_dict["exp"] = float_1 * float_2

for key in return_dict:
    print(return_dict[key])
```

2.7 The end is near but not here

Write a function that returns the next to last word in the string argument.

Words are any sequence of non-blank characters.

You may assume that the input string will always contain at least two words.

[ ]:
```python
def penultimate(string):
    lst= string.rsplit(" ", 1)
    return lst[-1]

print(penultimate("Hey hey doogie howser"))
```

2.8 Exclusive Or

The or operator returns a truthy value if either or both of its operands are truthy, a falsy value if both operands are falsy. The and operator returns a truthy value if both of its operands are truthy, and a falsy value if either operand is falsy. This works great until you need only one of two conditions to be truthy, the so-called exclusive or, also known as xor (pronounced "ECKS-or").

In this exercise, you will write an xor function that takes two arguments, and returns True if exactly one of its arguments is truthy, False otherwise.

take two operands, compare them. match/case statement holding state? No, even simpler. just a and not b

[ ]:
```python
def xor(a, b):
    if (a and not b) or (b and not a):
        return True
    return False

print(xor(5, 0) == True)
print(xor(False, True) == True)
print(xor(1, 1) == False)
print(xor(True, True) == False)
```

2.9 Odd Lists

Write a function that returns a list that contains every other element of a list that is passed in as an argument. The values in the returned list should be those values that are in the 1st, 3rd, 5th, and so on elements of the argument list.

Just a slice

[ ]:
```python
def oddities(lst):
    return lst[0::2]

print(oddities([2, 3, 4, 5, 6]) == [2, 4, 6])  # True
print(oddities([1, 2, 3, 4]) == [1, 3])        # True
print(oddities(["abc", "def"]) == ['abc'])     # True
print(oddities([123]) == [123])                # True
print(oddities([]) == [])                      # True

print(oddities([2, 3, 4, 5, 6]))
```

2.10 How Old is Teddy

Build a program that randomly generates and prints Teddy's age. To get the age, you should generate a random number between 20 and 100, inclusive.

[ ]:
```
import random

print(f'Teddy\'s age is {random.randint(20,100)}')
```

[ ]:
```
import random

name = input('What\'s the name of the person whose age you need?')
if not name:
    name = 'Teddy'

print(f'{name} is {random.randint(20,100)} years old!')
```

2.11 When Will I Retire?

Build a program that displays when the user will retire and how many years she has to work till retirement.

[ ]:
```
while True:
    current_age = int(input('What is your age?'))
    retirement_age = int(input('At what age would you like to retire?'))
    if type(current_age) == int or type(retirement_age) == int:
        break
print(f'It\'s 2024. You will retire in {2024+(retirement_age - current_age)} \n'
      f'You have only {(retirement_age - current_age)} years of work to go!')
```

2.12 Get Middle Character

Write a function that takes a non-empty string argument and returns the middle character(s) of the string. If the string has an odd length, you should return exactly one character. If the string has an even length, you should return exactly two characters.

[ ]:
```
def center_of(string):
    mid = len(string) // 2
    return string[mid - 1:mid + 1] if len(string) % 2 == 0 else string[mid]







print(center_of('I Love Python!!!') == "Py")    # True
print(center_of('Launch School') == " ")        # True
print(center_of('Launchschool') == "hs")        # True
print(center_of('Launch') == "un")              # True
print(center_of('Launch School is #1') == "h")  # True
print(center_of('x') == "x")                          # True
```

2.13 Always Return Negative

Write a function that takes a number as an argument. If the argument is a positive number, return the negative of that number. If the argument is a negative number, return it as-is.

```
[ ]: def negative(x):
         return (x * -1) if x > 0 else x

     print(negative(5) == -5)       # True
     print(negative(-3) == -3)      # True
     print(negative(0) == 0)        # True
```

```
[ ]:
```