Day 4: Functions and Dictionaries

July 13th 2023

verbal attendance today

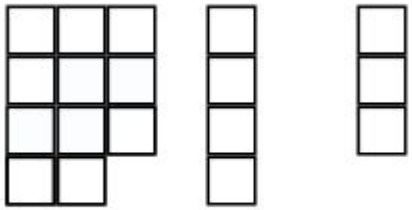
Modulo Math

- the symbol % represents **modulo** arithmetic
- result is also known as the **remainder** in division
- A modulo B (or A%B) means ‘after the maximum whole number of times B goes into A, what is leftover?’
  - the solution will **always** be less than B
  - the solution can be 0 – when would this happen?
- examples:

```
> 11 % 4
3
> 11 % 3
2
> 7 % 6
1
```

```
> 7 % 7
0
> 14 % 7
0
```

Modulo operation

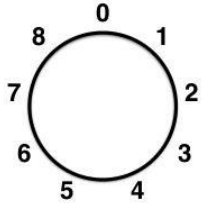


$$11 \; mod \; 4 \; = \; 3$$

ComputerHope.com

Modulus 9

0 mod 9 = 0	9 mod 9 = 0
1 mod 9 = 1	10 mod 9 = 1
2 mod 9 = 2	11 mod 9 = 2
3 mod 9 = 3	12 mod 9 = 3
4 mod 9 = 4	13 mod 9 = 4
5 mod 9 = 5	14 mod 9 = 5
6 mod 9 = 6	15 mod 9 = 6
7 mod 9 = 7	16 mod 9 = 7
8 mod 9 = 8	17 mod 9 = 8

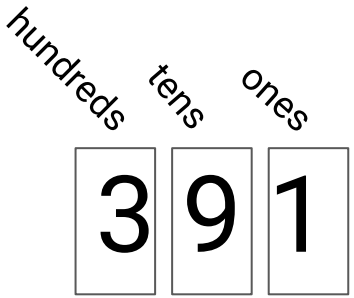


Modulo Example: Number Digits

how would we write code to return the hundreds, tens, and ones digit of a number?

start from ones digit – you will need it to solve for the tens digit (and you will need the tens and the ones to solve for the hundreds!)

hint: we will need both modulo (remainder) and division




```
num = 391
ones_digit = num % 10
tens_digit = ((num % 100) - ones_digit) / 10
hundreds_digit = ((num % 1000) - (num % 100)) / 100
print(ones_digit, tens_digit, hundreds_digit)
```

Warmup: Leap Year Problem

Write a script that prints whether a specified year is a leap year based on this rule:

✓	The year **must be evenly divisible by 4;**
✗	If the year can also be **evenly divided by 100**, it is ***not*** a leap year;
	unless...
✓	The year is also **evenly divisible by 400**. Then it ***is*** a leap year.

- Example of leap years: 2000, 2400
- Example of non-leap years: 1800, 1900, 2100

```
num = 391
ones_digit = num % 10
tens_digit = ((num % 100) - ones_digit) / 10
hundreds_digit = ((num % 1000) - (num % 100)) / 100
print(ones_digit, tens_digit, hundreds_digit)
```

Leap Year Problem

Write a script that prints whether a specified year is a leap year based on this rule:

```
## -------- leap year --------
year = 2000  #change the value
year1 = int(year % 4)
year2 = int(year % 100)
year3 = int(year % 400)
if year1 == 0:  # divisible by 4
  if year2 != 0:  # dont need to check 400
    print(True)
  else:  # is divisible by 100...
    if year3 == 0:  # also divisible by 400
      print(True)
    else:
      print(False)
else:  # not divisible by 4
  print(False)
```

```
year = 2000  #change the value
year1 = int(year % 4)
year2 = int(year % 100)
year3 = int(year % 400)
if year1 == 0 and (not year2 == 0) or (year3 == 0):
  print(True)
else:
  print(False)
```

almost always more than one solution to a problem!

Clarifications

- you do not have to submit the workspace (though you can!). we can see it live even without submission. it is a tool for in class, and is checked for participation (not that everything runs)
- homework **does** need to be submitted so we can officially grade it

FUNCTIONS

functions are like recipes for things you want to do lots of times but might have different values each time – for example, you might want to print text to your screen without writing how to convert inputs to pixels each time.

similar to writing a recipe for pizza – you might change the toppings or use a different kind of flour or sauce, but you will take the same steps with what you're given!

FUNCTIONS

It is important to note that functions and in general all code is sequential. So like recipes is important to follow a certain order. If you don’t follow the order it is possible that you end up burning the recipe or with buggy code.

you decide what your function accepts!

```
truenum = 934
val = input('what is your number?')
if val != truenum:
    print('not correct')
```

unexpected but successful!

```
truenum = 934
val = input('what is your number?')
if val <= truenum:
    print('too low!')
```

code fails!

FUNCTION NOTATION: Inside the function

```python
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

FUNCTION NOTATION: Inside the function

```
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

definition

FUNCTION NOTATION: Inside the function

```
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

name

FUNCTION NOTATION: Inside the function

```
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

inputs (aka parameters)

FUNCTION NOTATION: Inside the function

```
> min([1,2,3])
1
```

not all functions have multiple arguments or results! what does this function do?

you have also used the functions **len, str, int, input,** and **print**!

FUNCTION NOTATION: Inside the function

```
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

body

FUNCTION NOTATION: Inside the function

```
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

output(s)

FUNCTION NOTATION: Inside the function

```
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

comments / description

FUNCTION NOTATION: Inside the function

```
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

indentation!

FUNCTION NOTATION: Outside the function

```python
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

```
> myfuncname(1,2)
(3, 2)
> a,b = myfuncname(3,4)
> print(a)
7
> print(b)
12
> hello = 1
> world = 2
> myfuncname(hello,world)
(3, 2)
>
```

Our first function – counting the length of a list

- what should the input be? how many things do we want to take in?
- what should the output be?
- what code feature might we use?

Modulo Function 1: Flexible Number Digits

using the modulo code from before, write a function that takes in a number less than 10,000 and greater than 1 and returns the sum of the digits in the number

- you know the number will have between one and four digits, but you will need to check how many it has!
- once you have all the digits, **return** their sum

SCOPE

functions have a scope that is different from the scope of your whole program!

scope means “definition of variables”

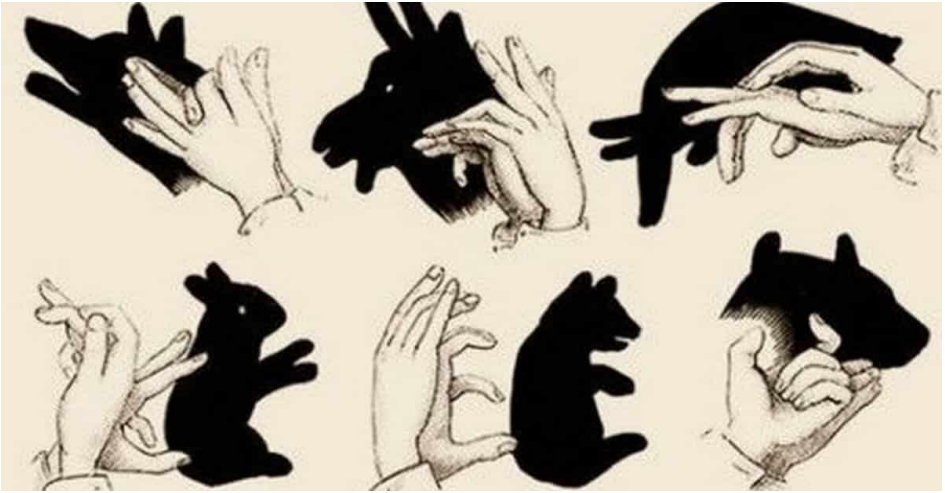
why would we want this?

example: pizza recipe

if you are making a pizza sauce and pizza dough, and read the instruction “knead the mixture,” how could this be confusing if you are following both recipes?

why would it be more helpful to see “topping” instead of “onion”?

SCOPE



scope is like what allows us to watch a puppet show and focus on the characters and story (a fox and a bird) and not on how they are made (paper, hands, etc)

outside and inside the function work with different information!

SCOPE

```python
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

```
> myfuncname(1,2)
(3, 2)
> a,b = myfuncname(3,4)
> print(a)
13
> print(b)
12
> hello = 1
> world = 2
> myfuncname(hello, world)
(3, 2)
```

environment diagram for the code we just ran!

SCOPE

```python
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

```python
a, b = myfuncname(2, 40)
print(a)
print(b)
print(input1_squared)
```

```
True
44
80
Traceback (most recent call last):
  File "main.py", line 26, in <module>
    print(input1_squared)
NameError: name 'input1_squared' is not defined
>
```

any variables you define inside your function are in **local** scope of that function – they live and die when that function is being used

if you don't return a value, it is lost forever!

goes both ways! you need to pass information to a function for it to use it

SCOPE

```python
def myfuncname(input1, input2):

    # returns the sum and product of two numbers
    # must take in two integers or floats!
    # strings will not work

    input1_squared = input1 ** 2
    result1 = input1_squared + input2
    result2 = input2 * input1
    return result1, result2
```

```python
a, b = myfuncname(2, 40)
print(a)
print(b)
print(input1)
```

```
True
44
80
Traceback (most recent call last):
  File "main.py", line 26, in <module>
    print(input1)
NameError: name 'input1' is not defined. Did you mean: 'input'?
>
```

any variables you define inside your function are in **local** scope of that function – they live and die when that function is being used

if you don't return a value, it is lost forever!

goes both ways! you need to pass information to a function for it to use it

SCOPE

```
def in1_bigger(input1, input2):
    return input1 > input2
```

```
> input1 = 3
> input2 = 1
> comp_result = in1_bigger(input2, input1)
>
> print(comp_result)
False
>
```

variables get “renamed” inside your function when they are passed into specific input parameters.

in a recipe, it will just call something ‘topping’ even if you know that your topping is spinach

common bug! don’t assume your names will hold inside the function. it only knows the **values** it is passed, not their names

why is comp_result false? 3 is larger than 1

SCOPE

at first, this might seem confusing. but it's very helpful to not have to know all the guts and variables in every function you use.

if you did, you couldn't name any variables what any other functions have used... it would be impossible to code!

this is also why you can pass variables or values to a function and it will work both ways

we have been using the range function without knowing start - step - stop this whole time!

```
> egg = 1
> salad = 2
> myfuncname(egg,salad)
(3, 2)
> myfuncname(1,2)
(3, 2)
>
```

SCOPE

area of caution: changing values of a variable

remember – inside the function does not know what outside names are

```
def increase_var(smallvar):
    # increases value of a variable by 5
    smallvar = smallvar + 5
    return smallvar
```

```
> smallvar = 10
> increase_var(smallvar)
15
> smallvar
10
>
```

```
def myfuncname(input1, input2):
    result1 = input1 + input2
    result2 = input2 * input1
```

local variable 'result2' is assigned to but never used
pyflakes

helpful message in replit to stop you from missing a return statement!

```
def square_number(x):
    square = x * x


sq = square_number(2)
print(sq)
```

Null Type – Functions That don’t return anything!

```
def sayhello():
    username = input('what is your name? ')
    print('it is nice to meet you, ' + username + '!')
```

```
> hi = sayhello()
what is your name? ratty rat
it is nice to meet you, ratty rat!
> print(hi)
None
>
```

code won’t throw an error, it is not wrong to not return anything! but putting it in a variable will give a **None** value

Homework Problem 1: Unit Conversion

Write 3 functions, each converting one unit to another:

1. A function `lbs2kg` which takes in a variable `lbs` (float) and returns the converted value in kg (1 lb = 0.453592 kg)
2. A function `feet2meters` which takes in a variable `feet` (float) and returns the converted value in meters (hint: 1 ft = 0.3048 m)
3. A function `inches2meters` which takes in a variable `inches` (float), and returns the converted value in meters (hint: 1 lb = 0.0254 m)

Homework Problem 2: Haley’s Diner

At Haley’s Diner, employees generally work only up to 40 hours each week at a certain rate, but when they have to work overtime (more than 40 hours), their rate increases by 50%.

For example, employee A gets $20/hr and worked 30 hours this week, the total pay is $600 : 30 * 20 = 600

Employee B also gets $20/hr but worked 50 hours this week, the total pay is $1100 : (40 * 20) + (10 * 30)

Write a function called `weekly_pay` which takes in two variables `hours` and `rate` and returns how much the employee earned as an output `pay`.

Homework Problem 3: Modulo Game

Preamble: In this problem, you will implement a popular game in South Korea called the 3-6-9 (sam-yuk-gu) game. In this game, you play with a number of people (the more, the better!) sitting in a circle. Then, people in the group take turns counting to a number, starting from 1, 2 and so on. However, when the number has a **'3'**, or **'6'**, or **'9'**, you do not say the number and clap instead. The number of times you clap will equal the number of times ' **3-6-9** ' occurs in that number. For example, you clap one time for **'6'** but clap twice for **'33'** and **'39'**. You will lose the game if you forget to clap correctly.

Instructions

- write a function **lets_play_369**(N) which receives the 'last number to be played'/total number of rounds as input **N**
- your function should count the number of claps happening in the game and return it as an output **count_clap**
- N is an integer less than 100
- you are not required to print out the game play record. However, it is highly recommended to visualize the counts while playing

Hint: How do you get the first digit and the last digit of any two-digit numbers using modulo operation and division?

An example of a gameplay with N = 13:

```
1,
2,
clap! (3),
4,
5,
clap! (6),
7,
8,
clap! (9),
10,
11,
12,
clap! (13),
```

>> Return : count_clap = 4

Homework Problem 4: Fibonacci Numbers

Write a function **fib** which takes in a number L (which must be at least 1) and prints out the fibonacci sequence of that many numbers

you will need to track how many numbers you have printed, as well as the values of the numbers you are working with in the sequence!

optional challenge: can you make the code print the sequence in one line?

The Fibonacci Sequence

1,1,2,3,5,8,13,21,34,55,89,144,233,377…

1+1=2	13+21=34
1+2=3	21+34=55
2+3=5	34+55=89
3+5=8	55+89=144
5+8=13	89+144=233
8+13=21	144+233=377

Environment Diagram (Optional Challenge)

set a to some number <20 and pick b = 3 or 4. how would you draw an environment diagram for a function that calls itself? hint: you will need multiple “environments”!

```
# --- recursion example
def mystery(a, b):
    if b == 1:
        c = a
    else:
        c = a + mystery(a, b - 1)
    return c
```

Homework Challenge Problem: What does this function do?

```python
def sum_digits(N):
    total_sum = 0
    if N < 10:
        total_sum += N
    elif N < 100:
        ones_digit = N % 10
        tens_digit = int((N % 100 - ones_digit) / 10)
        total_sum += ones_digit + tens_digit
    elif N < 1000:
        ones_digit = N % 10
        tens_digit = int((N % 100 - ones_digit) / 10)
        hundreds_digit = (N - ones_digit - (10 * tens_digit)) / 100
        total_sum += ones_digit + tens_digit + hundreds_digit
    else:
        ones_digit = N % 10
        tens_digit = int((N % 100 - ones_digit) / 10)
        hundreds_digit = ((N % 1000) - ones_digit - (10 * tens_digit)) / 100
        thousands_digit = (N - ones_digit - (tens_digit * 10) -
                           (hundreds_digit * 100)) / 1000
        total_sum += ones_digit + tens_digit + hundreds_digit + thousands_digit
    return total_sum
```