Arch 1, Week 1

Assignment: A1W1A1 - Year to month & day

Creation Date: 04-09-2023

What did I learn?

I learned nothing with this assignment.

How did I learn it?

Not applicable

Why/how did I solve it?

I wanted to have a variable for each calculation and since the input will be the year I've casted that to an int and called the variable 'y'.
When printing I used 3 loose print() methods to prevent one big line and to make the code a bit more readable. This is how it would have looked when I put everything on one line:

```
print("Years: " + str(y) + "Months: " + str(m) + ", Days: " + str(d))
```

Code Snippet

```
# Print what you need for the input
print("Enter amount of years:")

# Request input and convert it to an int so we can use it in a formula
y = int(input())

# Use the input (y) to calculate months and days
m = y * 12
d = y * 365

# Print the results - variable should be string since we concatenate it
to a string
print("Years: " + str(y))
print("Months: " + str(m) + ",")
print("Days: " + str(d))
```

Assignment: A1W1A2 - Tax & Tip

Creation Date: 04-09-2023

What did I learn?

I learned nothing with this assignment.

How did I learn it?

Not applicable

Why/how did I solve it?

Because I needed decimals and didn't want to write more code than necessary I cast the input to float and force the decimal that way.
Then when printing I use `%.3f` to say I want to print 3 decimals if the outcome of the calculations provide a 3 decimal outcome.

Code Snippet

```
# Request input and convert it to a float so we can use it in a formula
with decimals
price = float(input())

# Calculate tip, tax and total
tip = price / 100 * 15
tax = price / 100 * 21
total = price + tip + tax

# Print the results - Instead of converting the variable to string we
say that we expect a 3 decimal float
print("Tax: %.3f" % tax)
print(", Tip: %.3f" % tip)
print(", Total: %.3f" % total)
```

Assignment: A1W1P1 - Hello name

Creation Date: 04-09-2023

What did I learn?

I learned nothing with this assignment.

How did I learn it?

Not applicable

Why/how did I solve it?

The input method defaults to strings, so I concatenated the input which I have called name, to "Hello" in the print method to successfully pass the assignment.

Code Snippet

```
# Print what we need
print("What's your name?")

# Request input
name = input()

# Print input - no conversion needed for the variable as it's already a
string
print("Hello " + name)
```

Assignment: A1W1P2 - Year to month and day

Creation Date: 04-09-2023

What did I learn?

I learned nothing with this assignment.

How did I learn it?

Not applicable

Why/how did I solve it?

I wanted to have a variable for each calculation and since the input will be the year I've casted that to an int and called the variable 'y'.
When printing I used loose print() methods to prevent one big line and to make the code a bit more readable.

Code Snippet

```
# Print what we need
print("Enter amount of years:")

# Request input and convert it to an int so we can use it in a formula
y = int(input())

# Use input to calculate both months and days
m = y * 12
d = y * 365

# Print the result - variable should be string since we concatenate it
to a string
print("Months: " + str(m))
print(", Days: " + str(d))
```

Assignment: A1W1P3 - Room area

Creation Date: 04-09-2023

What did I learn?

I learned nothing with this assignment.

Why/how did I solve it?

I wanted to print what I needed separately from the input() method so I basically repeat the first `print()` and `int(input())` lines.
Then, since I want to calculate the area I just do `a = l*w` (meaning `area = length * width`) and print the result (`a`)

Code Snippet

```
# Print what you need
print("Enter Width:")

# Request input and convert it to an int so we can use it in a formula
w = int(input())

# Print what you need
print("Enter Length:")

# Request input and convert it to an int so we can use it in a formula
l = int(input())

# Calculate area with the width and height input we just gathered
a = l*w

# Print the result - variable should be string since we concatenate it to a string
print("The Area of the Room: " + str(a))
```

Assignment: A1W1P4 - Weight calculation

Creation Date: 04-09-2023

What did I learn?

I learned nothing with this assignment.

How did I learn it?

Not applicable

Why/how did I solve it?

I've created two variables (being `widget_height` and `gizmo_height`) as these values will never change and will be used globaly (hence why they are at the top).
I then followed up, two times, with the method `print()` and the method `input()` to determine how many widgets and gizmos we have.
When that's determined we calculate the weight total for the widgets and gizmos (`widget_total_weight` and `gizmo_total_weight`).
Then to fulfill the requirements of the assignment we add both **individual totals** to get the final total. Now if we print `total` we have fulfilled the requirements.

Code Snippet

```
# Widget and Gizmo height are static
widget_weight = 75
gizmo_weight = 112

# Request input and convert it to an int so we can use it in a formula
widget_amount = int(input())

# Request input and convert it to an int so we can use it in a formula
gizmo_amount = int(input())

# Calculate the total weight from both Widget and Gizmo
widget_total_weight = widget_weight * widget_amount
gizmo_total_weight = gizmo_weight * gizmo_amount

# Now add both weights so we have the total weight
total = widget_total_weight + gizmo_total_weight

# Print the total weight - variable should be string since we
concatenate it to a string
print("The Total Weight of the Order:" + str(total))
```

Assignment: A1W1P5 - Four digit sum

Creation Date: 04-09-2023

What did I learn?

I learned that you can convert a string to an array using

```
input_to_array = [char for char in input_to_str]
```

How did I learn it?

Not applicable

Why/how did I solve it?

To convert a string full of numbers (Ex: 1234) to a sum (1+2+3+4=10) I decided that it might be the best way to program
the software in 'steps'. Meaning that each variable, loop or statement will be limited in function but also always finish one task - an exception being the first variables which are needed to store data and therefore won't be doing any 'tasks' (i.e calculations).
The best way I could think of solving this is to treat the string as an array (in a loop) and then use `[]` to index that position.
I consider this a good approach because this way, again, you only have to do one 'task' to accomplish adding the total of the sum as well as actually 'writing' the formula.

To 'write' the sum we check in each iteration if the storage we are going to write to is odd. If it is odd we'll add a `+` sign first and then the number. If it is anything else then odd (that would be even) we only add the number.
This should be done because we never want to begin with a `+` sign but always want it after the first number for example.

Then, to actually calculate the final result we do a slight alteration/simplification of the code we made to write the sum and just convert the last number (in string form) to int and add that with `+=` to our last total from our previous iteration.
If you do this correctly, which I do, you should get both the right formula, as string, and the right total, as int which we need to convert to string again for the `print()`.

The way I create an array of the string is like this:

```
input_to_array = [char for char in input_to_str]
```

Code Snippet

```
# Request an input
input = input()

# Convert the input to string
input_to_str = str(input)

# Now we can convert the string to an array
input_to_array = [char for char in input_to_str]

# Create a variable so we can get the result of the sum later on
sum = 0

# Create a variable so we can store the formula (1+2+3+4)
sum_str = ""

# We create a counter so we know at which position in the string we are
counter = 0

# Loop through the formula which we converted to an array
for c in input_to_array:
    # If length of the formula we pass is odd we make sure to add a "+"
and then the next number
    if len(sum_str) % 2:
        sum_str += "+" + input_to_array[counter]

    # If the length of the formula is anything but odd we only put the
next number in
    else:
        sum_str += input_to_array[counter]

    # We always add the next number to the last total of sum until
    sum += int(input_to_array[counter])

    # Move one up so we can actually get the next number
    counter += 1

# Now that we are done with the loop we print both the formula and the
sum as one whole string
print(sum_str + "=" + str(sum))
```

Assignment: A1W1P6 - Hours, minutes and seconds

Creation Date: 04-09-2023

What did I learn?

I learned nothing with this assignment.

How did I learn it?

Not applicable

Why/how did I solve it?

I wanted to have a variable for each calculation and since the input will be the amount of days I've casted that to an int and called the variable 'd' to calculate the hours.
Once you have the hours you can calculate the minutes and once you have your minutes you can calculate the seconds.
After the calculations are done I simply printed the outcome with the variables casted to a string so concatenation is possible.

Code Snippet

```
# Print what we need
print("Provide an amount of days")

# Request an input and convert it as int so we can use it in a formula
d = int(input())

# Calculate the hours, minutes and seconds with the input
hours = d * 24
minutes = hours * 60
seconds = minutes * 60

# Print the results as strings. Variables need to be string because we
otherwhise concatenate an integer to a string
print("Hours: " + str(hours))
print(", Minutes: " + str(minutes))
print(", Seconds: " + str(seconds))
```

Arch 1, Week 2

Assignment: A1W2A1 - Immediate successor

Creation Date: 06-09-2023

What did I learn?

How Codegrade handles errors - I also overlooked relatively simple solutions in favour of more difficult ones which could be written with fewer lines. This resulted in spending more time on this assignment than I wanted.

How did I learn it?

I've looked at the errors which were outputted by Codegrade and compared them to the errors I got from my IDE (PyCharm).

Why/how did I solve it?

This was an interesting one for me. Not that is was a difficult assessment but because I needed to think simpler than I'm used to.
This is why my first thought was using the `datetime` library to get the current date, automatically detect when a month did or didn't have 31 days ect.
Although I did get far with that library Codegrade didn't like the exceptions the library was causing on the website - the issue wasn't present on my latop/pc.
Hence why I switched to my current version of the code (see below, under 'Code Snippet').

I used methods, why?: my main reason to use methods was to prevent that the code would become overwhelmingly long. A good example of this is the validation method, validate(). I could have changed the order of the code but since I was able to use methods I had the flexibility and therefore the possibility to validate the input more or somewhere else.
So besides it being useful for fewer lines of code it is also very useful if you are trying to debug quick and easy.

Considering you used a library first, why would you choose a library over your current solution?:
Well, a library tends to be programmed in a way it already checks a lot of edge cases for you which gives you more information when you need to debug your software. The interesting part though is that this reason is also the reason why Codegrade didn't accept it. - The most likely reason is that Codegrade forced certain prints which your local machine doesn't force. And since print() methods are expected to be the actual result of the assignment it didn't work.

So how does it actually work?:
- **main()** triggers the whole program
- **dateInput()** first I'm asking for user input and pass that to the `dateInput()` method.

This method is responsible for triggering the validation and then seeing if the validation succeeded or not. If the validation succeeded and everything is OK it will trigger the shift functions and print the shifted outcome. This would complete the assessment.
- **validate()** the validate method checks if the values which are passed are correct and if not it prints an error and returns `None`, if it is all good it will actually format the date again to guarantee the year, month and day are in the correct place (`YYYY-MM-DD`) and return it.
- **shiftYear(), shiftMonth() and shiftDay()** are responsible that the year, month or day can be both increased or decreased.

Code Snippet

```
# Validate year format
def validate(date_text):
    year = date_text[0:4]
    month = date_text[5:7]
    day = date_text[8:10]
    if int(year.replace('-','')) >= 2000:
        return f'{year}-{month}-{day}'
    elif int(year.replace('-','')) < 1900:
        print("Input format ERROR. Correct Format: YYYY-MM-DD")
    else:
        print("What happened?")

# Function to increase or decrease (aka shift) the year
def shiftYear(date_text, shift):
    year = int(date_text[0:4]) + shift
    return str(year) + "-01-01"

# Function to increase or decrease (aka shift) the month
def shiftMonth(date_text, shift):
    month = int(date_text[5:7]) + shift
    return date_text[0:4] + "-" + str(month) + "-01"

# Function to increase or decrease (aka shift) the day
def shiftDay(date_text, shift):
    day = int(date_text[8:10]) + shift
    day_str = str(day)
    if len(day_str) < 2:
        day_str = "0" + day_str
    return date_text[0:7] + "-" + day_str

# Execute our actions here
def dateInput(input):
    i = validate(input)
    if i != None:
        i = shiftDay(i, 1)
        print("Next Date: " + i)
        i = shiftMonth(i, 1)
        print("Next Date: " + i)
```

```
        i = shiftYear(i, 1)
        print("Next Date: " + i)

# Get input and pass it to the dateInput() function
def main():
    date_input = input()
    dateInput(date_input)

# Run the program
main()
```

An example of using the datetime library

```
# Define the dateInput() method
def dateInput(input):
    # Trigger the validate method
    x = validate(input)

    # If the validation is OK
    if x != None:
        # make the validated format a date of type datetime
        datetime.date = x

        # To shift one day into the future replace the current day with
current_day+1
        datetime.date = datetime.date.replace(day=datetime.date.day +
1)

        # Print our new date
        print(datetime.date)

        # To shift one month into the future and begin on day one of
the month
        datetime.date = datetime.date.replace(month=datetime.date.month
+ 1, day=1)

        # Print our new date
        print(datetime.date)
```

Assignment: A1W2P1 - Even or Odd

Creation Date: 07-09-2023

What did I learn?

Nothing

How did I learn it?

Not applicable

Why/how did I solve it?

I requested user input, casted it to int and then did modulo (%) with 2 to see if the remainder is uneven.
If the remainder is not 0 then the result is uneven, if not the result is even.

Code Snippet

```
# Ask for input and immediately make it an int
i = int(input())

# Modulo to check if the input is even or odd as it will check the remainder
if i % 2:
    print("Odd")
else:
    print("Even")
```

Assignment: A1W2P2 - Leap year

Creation Date: 11-09-2023

What did I learn?

I learned how to calculate a leap year

How did I learn it?

I looked up the formula

Why/how did I solve it?

I requested user input and casted it to float. I've casted it to float because the outcome of the leap year can have decimals.
If we have the float we will divide it by 4 to see kickstart the calculation. Then if that result is odd it can't be a leap year, and if it is even it is (most likely) a leap year.

Code Snippet

```
# Get input
i = float(input())

# Divide by for
leap = i / 4

# If year is even after dividing it's a leap year, if not it is not a
leap year
if leap % 2:
    print("not leap")
else:
    print("leap")
```

Assignment: A1W2P3 - Sides to shape

Creation Date: 11-09-2023

What did I learn?

I already knew it

How did I learn it?

I already knew it

Why/how did I solve it?

I requested user input and casted it to int. Once that was done I could use the `nr` in `shapes[]` as an index.
If that is done we can print `shapes[nr]` as `shape`.

Code Snippet

```
# List all kind of shapes
shapes = ["", "", "", "triangle", "quadrilateral", "pentagon",
"hexagon", "heptagon", "octagon", "nonagon", "decagon"]

# Get the number of the shape so we can index is later
nr = int(input())

# Request the index of the shapes as variable shape
shape = shapes[nr]

# Print shape
print(shape)
```

Assignment: A1W2P4 - Triangle type

Creation Date: 11-09-2023

What did I learn?

Nothing I already knew

How did I learn it?

Not applicable

Why/how did I solve it?

I requested user input and then checked through if and else if, if the sides are, or are not, equal to each other.
Based on if and how many sides are equal it will print what kind of triangle it is.

Code Snippet

```
i = input()

a = i[2]
b = i[7]
c = i[12]

if a == b and a == c:
    print("equilateral triangle")
elif (a == b and not a == c) or (a == c and not a == b):
    print("isosceles triangle")
elif (a != b) and (a != c):
    print("scalene triangle")
```

Assignment: A1W2P5 - Dutch holidays

Creation Date: 11-09-2023

What did I learn?

I didn't know `json.get(key)` was a built-in function in python.

How did I learn it?

I've looked up how python uses JSON and if I needed a library for it or not.

Why/how did I solve it?

I request user input, define a table of Dutch festivities with the `month and day` as key and the name of the festivity as value.
Then I use `festivities.get(input)` to see if there is a key based on the provided input. If the input is equal to an existing key it prints the name of the holiday.
If not, it will print "does not exist".

Code Snippet

```
i = input()

festivities = {
    "month 1, day 1": "Nieuwjaarsdag",
    "month 4, day 7": "Goede Vrijdag",
    "month 4, day 9": "Eerste Paasdag",
    "month 4, day 10": "Tweede Paasdag",
    "month 4, day 27": "Koningsdag",
    "month 5, day 5 ": "Bevreidingsdag",
    "month 5, day 18": "Hemelvaartsdag",
    "month 5, day 28": "Eerste Pinksterdag",
    "month 5, day 29": "Tweede Pinksterdag",
    "month 12, day 5": "Sinterklaas",
    "month 12, day 25": "Eerste Kerstdag",
    "month 12, day 26": "Tweede Kerstdag",
}

if festivities.get(i) is not None:
    print(festivities[i])
else:
    print("Does not exist")
```

Assignment: A1W2P6 - Dog years

Creation Date: 11-09-2023

What did I learn?

Nothing I already knew

How did I learn it?

Not applicable

Why/how did I solve it?

First of all I define (global) variables with one of them requesting user input.
Then I check if the user input is a negative number. If it is lower then a 0 it will print an error and exit the program.

If it is a positive number it will enter a while loop and only exit once the iteration value (which is initially 0). In the loop we check if the iteration is smaller then 2 which, in other words, checks if the dog is younger than 2. If it is younger it will add 4 dog years, if it is either equal or bigger than 2 it will add 10.5 dog years. When the additions are done we'll increment the iteration value with `+= 1` so we won't stay in the loop forever.
Once the loop is done we'll print the final age.

Code Snippet

```
i = int(input())
iteration = 0
age = 0
year = 0
dog_years_normal = 4
dog_years_first_two = 10.5

if i < 0:
    print("Only positive numbers are allowed")
    exit(1)
else:
    while iteration < i:
        if iteration < 2:
            age += dog_years_first_two
        else:
            age += dog_years_normal
        iteration += 1
print(age)
```

Assignment: A1W2P7 - Chessboard colors

Creation Date: 11-09-2023

What did I learn?

I learned that `.lower()` and `.upper()` work within Python.

How did I learn it?

I already knew it existed in other programming languages, so I tried it in Python as well and it worked.

Why/how did I solve it?

I basically solved it by counting columns and rows which basically creates a grid and then because we invert colours each row we can use modulo to see if we have an even or uneven number. The same goes for the columns.

First I do check if the type of column is a string, and the type of row is an integer. This is how we can confirm that we correctly separated the algebraic notation.
When we did it correctly the column part will lower the whole alphabet string, loop through each position of the string and see if the index of the current iteration is equal to the column which we seperated from the algebraic notation.
If the index is equal to the column we exit the loop, otherwise we add one to the column count, so we know we are not in the right column yet.

Almost the same goes for the rows, but this is a bit easier. In this case we enter a while loop instead of a for loop and we add one row (`r_count += 1`) as long as the row count is lower then the row (the number we separate from the algebraic notation).

When we did that we want to print either black or white, or white or black (since we need to invert the colours each time).
The way we could solve this is to see if the colum count, but with an addition of one, is odd.
The reason that we add one is that we want to prevent to do modulo on a zero.
Now we did that we just simply check if the row is odd or not and then print the colour.

Now we are able to print black and white correctly based on algebraic notation.

Code Snippet

```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
c_count = 0
r_count = 0

i = input()
column = i[0].lower()
row = int(i[1])

if type(column) == str:
    for a in alphabet.lower():
        if a == column:
            break
        else:
            c_count += 1

if type(row) == int:
    while r_count < row:
        r_count += 1

if (c_count + 1) % 2:
    if (r_count % 2):
        print("black")
    else:
        print("white")
else:
    if (r_count % 2):
        print("white")
    else:
        print("black")
```

Assignment: A1W2P8 - Licence plate

Creation Date: 13-09-2023

What did I learn?

I've learned how `license.split()` and `.isnumeric` works in Python.

How did I learn it?

I looked up potential solutions to easily get the data from the license plate number, without retrieving the dashes.

Why/how did I solve it?

Since there is a whole list of patterns which needed to be marked as valid I began with a few patterns and seeing if my solutions worked. I constantly checked my changes as well, so I exactly knew when the script did or didn't know when a pattern was valid.

In the beginning, before I used methods, to mark if the license plate was correct I made use of a variable called `good` which would be set to `1` (true) so I could call the variable later and check if the license plate was in fact passing the validation.
The code would look something like this:

```
# Checking if the license plate is the length it should be
if license_plate_length == 8:
    # Are there dashes as 3rd and 6th character?
    if license_plate[2] == "-" and license_plate[5] == "-":
        # Are the first two characters the same as the 4th and 5th
character?
        if (license_plate[0] + license_plate[1]) == (license_plate[3] +
license_plate[4]):
            # Are the last two characters NOT the same as the 4th and
5th character?
            if (license_plate[6] + license_plate[7]) !=
(license_plate[3] + license_plate[4]):
                # Then this license plater pattern is valid
                good = 1

    # Print if the lincese plate is valid
    if good:
        print("Valid")
    else:
        print("Not Valid")
```

Eventually I switched to methods and made life a bit easier to validate license patterns. I think I've cut my code in half with using the `.split()` and `.isnumeric` method.

The `split("-")` method was a true lifesaver because it removes the dashes and makes a list for you. The dash basically says where the comma should go for the list. (So `XX-XX-99`

becomes `[XX, XX, 99]`). When you can call them with an index and use `.isnumeric` you cal also see if that stored string *only* contains numbers. This is really useful to because we can now detect where the '99' is from 'XX-XX-99'. If you make variations on this with the `len()` method as well we suddenly can detect variations as 'XXX-99-X' as well.
See the final code below.

Code Snippet

```
def validate_license(license):
    # Split into parts
    parts = license.split("-")

    # Validate format
    # Check if the license plate is long enough
    if len(parts) != 3:
        return False

    # We need to have a numeric value in our first and last 'part' if
there are any numerics
    if parts[2].isnumeric() != parts[0].isnumeric():
        return False

    # Checks for patterns:
    # XX-99-99 and 99-XX-XX
    if parts[1] == parts[2] and parts[0] != parts[1]:
        return True

    # Checks for patterns:
    # 99-99-XX and XX-XX-99
    if parts[0] == parts[1] and parts[0] != parts[2]:
        return True

    # Checks for patterns:
    # 99-XX-99 and XX-99-XX
    if parts[0] == parts[2] and parts[0] != parts[1]:
        return True

    # If every part is unique
    if parts[0] != parts[1] and parts[0] != parts[2]:
        # Checks for patterns:
        # 99-XXX-9, XX-999-X and XXX-99-X
        if (len(parts[0]) + len(parts[1])) == 5 and len(parts[2]) == 1:
            return True

        # Or check for (the reversed) patterns:
        # 9-XXX-99, X-999-XX, 9-XX-999
        if (len(parts[1]) + len(parts[2])) == 5 and len(parts[0]) == 1:
            return True
```

```
def _main_():
    # Request user input
    license = input("License: ")

    # Pass the user input to the method, so we can validate it and
immediately check  if we get True or False back.
    if validate_license(license):
        print("Valid")      # True
    else:
        print("Invalid")    # False

# Run the script
_main_()
```

Arch 1, Week 3

Assignment: A1W3A1 - Predefined templated

Creation Date: 17-09-2023

What did I learn?

Not applicable

How did I learn it?

Not applicable

Why/how did I solve it?

As Cigdem Okuyucu said this to me; I won't specify how I solved it or what my code is if I didn't learn anything new in the assignment/problem.

Code Snippet

Assignment: A1W3P1 - Simple palindrome

Creation Date: 18-09-2023

What did I learn?

I learned that you can join a list like this: `x = ''.join(myDict)`.

How did I learn it?

I've looked up easy and short solutions and wanted to do this with a list.
And, in my case, I needed to convert the list back to a string with join.
I didn't want to use something like this `x = mySeparator.join(myDict)` so I looked up a solution which is a tiny bit shorter and 'easier' to use.
I found and settled on `''.join(reversed_i)`. This is actually a really obvious solution as you replace the variable `mySeperator` with a string. So in core it basically works the same.

Why/how did I solve it?

The list is reached by casting the input (string) to a list like this:

```
i = input()
reversed_i = list(i)
```

And then reversing and stringifying the list like this:

```
reversed_i.reverse()
reversed_i = ''.join(reversed_i)
```

Code Snippet

```
# Ask for user input
i = input()

# characters to filter.
punct_marks = ",.?!;"

# Remove all characters listed in 'puct_marks'.
count = 0
for p in punct_marks:
    i = i.replace(p, "")
    count += 1

# Reverse filtered input
reversed_i = list(i)
reversed_i.reverse()
reversed_i = ''.join(reversed_i)

# Check if the input is the same as the reversed input
result = ""
result = f'{i} is '
if i == reversed_i:
    result += "a palindrome"
else:
    result += "not a palindrome"

# Print if the input is a palindrome or not
print(result)
```

Assignment: A1W3P2 - Advanced palindrome

Creation Date: 18-09-2023

What did I learn?

Not applicable as the code I used is basically the same as the code I made for `A1W3P1 - Simple palindrome`.
The only difference is that the result print is expected to be different and I needed to add one extra `.replace()`. Specifically this replace `.replace(" ", "")` so it actually works with sentences.

How did I learn it?

Not applicable

Why/how did I solve it?

As Cigdem Okuyucu said this to me; I won't specify how I solved it or what my code is if I didn't learn anything new in the assignment/problem.

Code Snippet

```
```

Assignment: A1W3P3 - Modular rectangles

Creation Date: 18-09-2023

What did I learn?

Not applicable

How did I learn it?

Not applicable

Why/how did I solve it?

As Cigdem Okuyucu said this to me; I won't specify how I solved it or what my code is if I didn't learn anything new in the assignment/problem.

Code Snippet

Assignment: A1W3P4 - Celcius to Fahrenheit

Creation Date: 18-09-2023

What did I learn?

Not applicable

How did I learn it?

Not applicable

Why/how did I solve it?

As Cigdem Okuyucu said this to me; I won't specify how I solved it or what my code is if I didn't learn anything new in the assignment/problem.

Code Snippet

Assignment: A1W3P5 - Multiplication table

Creation Date: 20-09-2023

What did I learn?

I've learned what the enumerate function does and how I can use it in my code.
It returns a tuple with the first value as the count and the second one as the actual value from the list you're enumerating.
So if you use the following:

```
# Our list
list = ["item1", "item2", "item3"]

# Enumerating list and return a tuple
for count, item in enumerate(list):
    # Then print the count and the item
    print(count, item)
```

The output from the code above would be:

```
0 item1
1 item2
2 item3
```

The count will start at 0 and end when enumerate is finished enumerating through the list.

How did I learn it?

I've looked up easy and short solutions which I could use in this and future assignments.
Then I found what `enumerate()` does and how I could use it.

Why/how did I solve it?

I solved the assignment by making three different list and filling these with numbers by using a for loop with a specific range.
The `column(_joined)` variables are responsible for the right numbering while the rows actually does multiplication.
Then, at the end, once all variables are ready (so when all their respected values are added) we can enumerate on the rows (which we put in a list called `tables`).
Because of enumerate we can also count up with each print. This way we get the final output:

```
     1    2    3    4    5    6    7    8    9    10
1    1    2    3    4    5    6    7    8    9    10
2    2    4    6    8    10   12   14   16   18   20
3    3    6    9    12   15   18   21   24   27   30
4    4    8    12   16   20   24   28   32   36   40
5    5    10   15   20   25   30   35   40   45   50
6    6    12   18   24   30   36   42   48   54   60
7    7    14   21   28   35   42   49   56   63   70
8    8    16   24   32   40   48   56   64   72   80
9    9    18   27   36   45   54   63   72   81   90
10   10   20   30   40   50   60   70   80   90   100
```

Code Snippet

```python
# Define lists
column = list()
row = list()
tables = list()
# Create columns
for n in range(0, 11):
    if n != 0:
        column.append(str(n))
    column.append("\t")
# Create rows
for n in range(1, 10):
    row.append(str(n) + "\n")
# Create tables
for i in range(0, 11):
    row = ""
    # Row 1
    if i == 0:
        for j in range(1, 11):
            row += str(j) + "\t"
    # All other rows
    else:
        for j in range(1, 11):
            row += str(j * i) + "\t"
    # Add the row
    tables.append(row)
# Join lists so they both become a string
column_joined = "".join(column)
row_joined = "".join(row)
# Print each index (i) with their value (table) from the list (tables)
for i, table in enumerate(tables):
    if i < 1:
        print(f"\t{table}")
    else:
        print(f"{i}\t{table}")
```

Assignment: A1W3P6 - Binary to Decimal

Creation Date: 20-09-2023

What did I learn?

Not applicable

How did I learn it?

Not applicable

Why/how did I solve it?

As Cigdem Okuyucu said this to me; I won't specify how I solved it or what my code is if I didn't learn anything new in the assignment/problem.

Code Snippet

Assignment: A1W3P7 - Truth Tables

Creation Date: 20-09-2023

What did I learn?

Not applicable

How did I learn it?

Not applicable

Why/how did I solve it?

As Cigdem Okuyucu said this to me; I won't specify how I solved it or what my code is if I didn't learn anything new in the assignment/problem.

Code Snippet