Week 0 - functions, variables 21/6/2025

CS50 Python

- code hello.py → Python's extension
  - ↳ VS code command used to open/create a file, folder, etc.

.python hello.py Interprets code into bina
- ↳ "Interpreter": Program that does this process!

Built-in functions:

print("text") → output

input("Displayed text") → Input with a msg (Returns str)

Single line Comment

""" Multi-line comment """

print()
- Concatenation: print("text" + var)
- Passing Argument: print("text", var) → Auto-inserts a space before the variable (Separator)
  - ↳ Can be multi-Argument
  - sep=' ', end='\n' (Def. values)

Parameters vs. Arguments

Parameters: what can be passed into a function

Arguments: The actual values passed into the function (cha

str → String (datatype in Py.)

↳ Holds functionalities: .str()

.capitalize()

Single Quote or Double Quote? → As you like! But be consistent!
- "Python DOGS"
- "David Malan"

Positional Parameters vs. named Parameter

Based on order

ex: print("text")

1st Param!

also have a position, but can be called explicitly

ex: print("text", end="?")

called explicitly!

\ → Backslash: Escape character/Sequence

ex: \", etc.

print(f"hello, {variable}") "format"

↳ most used one!

Python's interactive mode → simply in CLI write 'python' (write code that's interpreted immediately)

int() → Convert to int data Type

float() → //float//

round() → To nearest integer

variable:, → formats No. with Comma

1,000

How to invent your own function?

def funName(): → All the following indentation will be included
f define .... #Code Here → parameters → You can add default values as well if needed fun(name="")

4 spaces = Tab

return value/variable

*A function must exist before it's called (At the top)

⚠ def main(): → convention to place code inside it + Any order of functions will be allowed

Last thing to call in file → main()

*Scope of variables (local/global) exists in Python

n**2 → n^2 (raise to power of 2) OR pow(n,2) (So many ways to solve same problem)

PSet 0 - Notes

- Utilize Python's Documentation & Duck Debugger (CS50's) to learn correctly throughout the course! (To reap benefits)
- chaining functions/methods is allowed ex: lower().replace()
- functions learned: .lower() .replace(old,new) .removePrefix("char"/string) .removeSuffix("//")
  - .lower() → convert entire string lowercase
  - .replace → replace a substring
  - .removePrefix / .removeSuffix → substr - Remove when know it's begin/end
- str() can be used to ..., just like int() & float()

22/6/2025

WEEK 1 - Conditionals

if condition: → "Boolean expression"
#Code to execute

- If it's a simple condition () are not necessary [syntax]

⚠ Python doesn't have switch cases (unlike C++ & Java) → has "match" similar to it (literal difference is keyword)

elif condition: → Used to check as alternative if previous
else: conditions weren't met (Recall flowchart)

Keywords in Python: or , and → separate Boolean expression

Chaining comparison operators: if 90 <= score <= 100:
(Possible in Python, not in C++ & Java)

% modulo operator → returns remainder after division of 2 numbers

bool → data types that include: True or False (notice capitalization)

match variable: ⟶ switch case in Python ⚠ Doesn't require Breaks, _ instead of default (like in other langs)

```
case value:
    #handle
case value1 | value2 |... :
    #handle multiple values (single Bar)
case _ :
    #handle unknown conditions (underscore)
```

Pset ① - Notes

str.split(" ") → splitting character
→ splits a string into a list based on the seperator parameter (so you can access substrings individually)
→ returns a list (e.g array)

str.find("substring") → returns index of 1st occurence, otherwise -1 [useful in Boolean expressions]

Formatting decimal output → Print(f"{variable:...f}" + f"formatted string again!")
+ more formatted strings! ↳ syntax for a single float

⚠ Arithmetic operations between a string & float/int: ~~error~~ error or unexpected output (convert 1st)

str.lstrip(" ") → strips/removes a leading(only) character/s (look up syntax)

23/6/2025

Week 2 - Loops

⚠ Python supports i+=1, but not i++

```
while condition:
    #Code
```

list → Data Type in Python

ex: [0,1,2] → square bracket syntax

```
for i in range(No.):
    #Code
```

→ Built-in function that literally returns a sequence of the specified number times.

keywords — any temp. var. name — can be a list aswell! ex: for i in [1,2,3,4]:

standard: . If you're not gonna use for loop's variable; Pythonic way: for _ in list:
↳ underscore

Interesting Pythonic feature: print("meow" * 3) → imitates loops!

To validate input standard way → while True: # input & break if valid

Hey world, just like continue

variables defined in a loop are still within the function's scope → unlike other languages.

ex: def test():
    while True:
        n = input()
        break
    return n # VALID in Python

How to declare a list & access elements?
list = [1, 2, 5]
list[0] list[1] list[2] → Indices Access syntax

len(list) → returns length of list

If you need the index no. while for looping, access the list through indices rather than the automatic object

dict → "dictionaries" another data type in Python e.g. a data structure
↳ key value pairs

How to declare a dictionary & Access elements?

words = { → curly braces
    "Hello": "world", → comma seperated
    "Goodbye": "cs50"
}
↳ key ↳ value

on diff. lines for aesthetics

words["Hello"] Accesses key called "Hello" & get's it's corresponding value "world"

⚠ lists indices are numeric
⚠ dictionaries keys are what matter!

How to iterate over dicts?

for student in students:
    print(student, students[student])

only the key — used the key to access the value!

List of dictionaries syntax: (similar to objects)

students = [
    {"name": "Ro", "age": "18"}, 1st dic.
    // // 2nd
    ..
]

special keyword Python: None (capitalized)
↳ Absence of a value, just like Null!

dictionaries → very high performance & function-rich
Python recommends snake case not camelCase
first_name vs. firstName

why use functions? Abstraction, Consistency, easier fixes, etc.

Part ② - Notes

char or string.isupper() .isalpha() .isdigit()
uppercase alphabet number

exists?
check variable inside a list → if var in list
check key inside a dictionary → key in dict
(returns bool value) (same syntax)

abs(n) → returns absolute value of a Nb.

len() → can also be used on strings ("list of chars")

WEEK 3 - Exceptions "Problems/Smth has gone coorng"

"Literal"
↳ something user types in
(common in error msgs)

SyntaxError → your mistake, didn't follow Python's language Rule (syntax)

ValueError → Invalid operation/conversion, input, etc. (a value) "base 10" → Decimal system
(Default system Python use

How to handle errors? try/except → Make Precisely handle "Run-time" errors

```
try:
    #code
except ErrorType:
    #handling code
```

⚠ Bad Practice: Generic error type instead of a specific type
↳ why? May hide bugs & indicates laziness

NameError → A variable, function, etc. error (not defined, ...)

```
try: #code
except ErrorType: #code
else: #code
```

Additional feature in Python: 'else'
→ only executes if an exception doesn't happen / is raised

Python's ~~global~~ scope: different scopes for diff. functions, classes, or modules
↳ If it's the same, then that variable will be inside that function's scope

keyword: "Pass" → used when an exception is caught, but you want to 'Pass' on doing Anything with it e.g ignore it

Pythonic way unlike other languages → try x exception are preferred more than conditionals (both are acceptable)

not → keyword (!) raise ExceptionType → raise an exception for the caller to handle!

Post(3)-notes

str.title() → Title cases a string: "hello world" to "Hello World"

EOFError → Ctrl-d, common way of ending input to a program

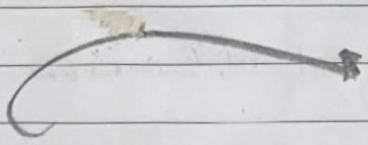
round(number, decimalplaces)
↳ optional (if not stated, the fraction part 0. will not be displayed)

*overheated → Hardest post to all a while
↳ what Helped: solving 1st on pen & paper & understanding the problem
+ Alot of functions created to make code Abstract!

To Pad with zeros: f"{variable:023}" ex: 9 becomes 09
↳ Has to be int

List
.index("value") → finds index of a value inside a list

Instead of utilizing indices in split
ex: list = date.split() → easier way → day,month,year = data.split()
and access list[0],..



WEEK4 - Libraries

25/6/2025

modules → Library in Python: to encourage Reusability & sharing code (Across your Project or other Projects)
↳ contains one or more functions

import modulename → loads the content of the module into your file [every-thing]

vs.

~~import~~ from modulename import functionname, ... → loads a specific choice into your namespace/scope of file (as/if needed)

Both ways have diff. use cases & are valid

ex: 1st way: random.choice() — lies in the module's scope
2nd way: choice() — lies in our file's scope [used immediately]

random.shuffle(list) → Actually shuffles the argument, doesn't return a new list [unlike a lot of what we've seen; why?]

statistics → another good ~~library~~ module

Command-line-Arguments → when Running through a command line environment

sys: module → sys.argv → list of the command you typed [0] → filename.py
→ No arg, just len()
→ sys.exit("msg") → exit Program

How to make a multiple words in a single location? Python word.py "Yo Hesham"
*Quotations

How to access a subset (slice) of a list

Syntax: list[startIndex : endIndex]
necessary ↳ optional

Package → A module implemented in a folder [3rd Party Libraries]
↳ Includes multiple modules & sub packages!

Where to get Packages? → PyPI (Python Package Index) (pypi.org)] just use pip!
Pip → Python's Package Manager (Auto-downloaded with Python)
Command e.g. Pip install packagename

Pip vs npm → installs packages in a local node_modules [Project specific]
→ confusing default behavior
→ installs packages globally (system wide)

APIs → 3rd party services or local really that you request (`requests` package) → imitates a browser allows internet requests
→ little data → HTTP requests through Python

JSON (JavaScript object notation) → "language agnostic format" for exchanging data between computers
independent of any specific programming language (not JS only) → "not lang. specific"
→ text-based & key value pair syntax

HTTP req.
API URL

How to request? `response = requests.get("URL")`
`data = response.json()` → parses json content, and returns a corresponding Python object [dictionary]

How to format the .json response? Package: import json
(to look readable) ex: `json.dumps(response.json, indent="2")`

Custom libraries → make your own libraries

How? locally: simply create .py file & import it into another file! → define functions inside of it

However, a problem arises if you unconditionally call main() in the imported file, it'll be executed when running the current file: How to avoid?

`__name__` → special Python variable: Auto-set by Python whose value is set to be "main" when a file is run through command line

`if __name__ == "__main__":`
`main()`
} only run main() for the target file!
(ignores other modules as `__name__` inside other modules/file will not be `__main__`)

shorts

Package: collection of multiple modules. How to create one? ① create a folder (will be ur package n.)
② place modules inside
③ create '__init__.py', that's empty!
tells/helps Python tell the folder is a package

/museum → package name
ex: museum.artists.py → module → How to acess? from museum.artists import funcname

Code style: PEP 8 (Provides style standards in Python community) (style guide)

Popular Code formatters: 'black' → pip install black → black filename.py (done!)

Post 4 - Notes

*If you're expecting a datatype specifically returning from a func() → explicitly convert ex: int(input) *so Error in Case

List = ["...", ..., .]
if value not in ["in", ..]] not in is valid (English lol)
if value not in list

How to add a new elem. to a list? list.append(item)

Also like placing repeated user input into functions: get_level()
→ good practice tbh

Accessing a key within a value
data["name"]["address"]
gets a value, which has a key called address

f"{amount:,.4f}
↳ separates No. for readability with commas

request.RequestException → catch with requesting APIs

WEEK 5 - UNIT TESTS 26/6/2025

• Better practice: using code that tests your own code: rather than manual input or others like check50

Convention: test_testfilename.py → Purpose: test functions (or program) inside the target file
ex: cal.py
How? import from target file like it's a module!

function convention: test_functionName() → Purpose: Automate testing!

assert(BooleanExpression) → Has to be True/Correct, returns specific line & AssertionError
like if () not returned →

* There are more tools → this is really simple

Pytest → 3rd Party Program That Automates testing of code: You write the tests (install with pip)

"unit testing" → testing individual units (like functions) of your programs

→ Handles all the tries/except/printing (All standards for testing) → allows you to focus on testing cases rather than focus on complex code!

ex: without Pytest

```
imports
def test_fun():
    try:
        assert(//)
    except AssertionError:
        #Handle Printing

def main():
    test_fun()

main() → with if __name__ etc
```

vs. with Pytest

```
imports
def test_fun():
    assert(//)
```

→ only create simple test func with no main

How to run? Pytest filename.py

Done!

* Really helpful practice → To have your code Byte-sized functions so they're testable!

Current Problem with Pytest? → Don't see result of other cases if one can't on is error

Solution: categorize test functions, rather than a big one

So a summary of diff. functions I wrote!

ex: +ve, -ve, zero function each

How to check if a case raises an exception? → with pytest.raises(ErrorType):

→ from pytest

test case → No need for assert

* Good Practice: writing functions that return values & avoid side-effects → like Print()

→ why? So they can be testable

How to test a folder of test files? ① __init__.py → treats folder as a package

② Pytest foldername

Major adv. for writing tests → if a teammate edits, it ensures they haven't broken anything!

WEEK 6 - file I/O

27/6/2025

sorted(list) → returns a sorted list

file = open("file.txt", "mode")
file.write("var")
file.close()

"w" → write
"r" → read, "a" → append
↳ Default (no need to specify)

Pythonic way:

```
with open("file.txt", "mode") as file:
    file.write(var)
    # No need to close file!
```

file.readlines() → returns list of all lines OR for line in file → more clear than func()
(either can)

once you're done with file operations → only have code related to it within 'with' block

csv → comma seperated values (multiple values in each line)

Given a dictionary or list: how to sort by key?

sorted(list, key = "value", reverse = False) → list
sorted(dict, key = func, reverse = True) → dict
↳ function which key to sort

better with lambda ← ex: get_func(): return dict["key"]
↳ use the return value by sorted() to determine order

Anonymous functions / Lambda function → when function is not called again / needed

syntax: lambda paramName : returnvalue
↳ if multiple params, seperate by comma (,)
ex: lambda student : student["name"] → returns value of the student's name key

---

import csv → Handles all csv seperation & edge cases

reader = csv.reader(file)
→ Has a list of lists ← comma seperated values!

csv.DictReader → in csv file add Title1
ex: name, home
data1, data2 ...
→ Access each row as key like a dict
→ Has a list of Dicts

csv.writer & DictWriter → .writerow(dict)
↓
.writerows(list)

However, when declaring csv.DictWriter(file, fieldnames = [list])
↓
headers in csv

*I do believe DictWriter/Reader is better

PIL → "Pillow" 3rd Party library for operations on image-files

Best@-sides

- Cleaner way to check if a file exists (try catch in main, leave function clean
  - FileNotFoundError
- reader itself is an object; don't pass it & expect to be treated as a list!
- list(iterable) → creates a list object
  - better than manual appending
- os.path.splitext(str) → returns list with [file name, .extension]

only use csv's library when dealing with csv files

str.endswith(" ")] more useful than
.startswith(" ")] using .split sometimes

.isspace() → bool (content is all spaces (whatever), false if empty)

csvwriter.writeheader() → writes field names as per csv
.writerow(s) (dict or list) → ex ({"first": value, ...})

10/7/2025

WEEK 7 - Regular Expressions "regexes" → a pattern! / Patterns

import re → very helpful built in functions

used in defining the patterns

- . any character
- * 0 or more repetitions
- + 1 / / //
- ? 0 or 1 repetition
- {m} m repetitions
- {m,n} m-n repetitions

"edu" → these 3 specifically!
To use . character literally use Escape Sequence

example: re.search(".+@.*", variable)
any char, 1 or more any char, 0 or more

re.search(".@", variable")
↳ any character! (only one)

*Regular expressions can be used in VS Code & Google Forms!

^ match start of the string

$ match end of the string

ex: "^@$"
exact match of @

However, a "raw" string is required (indicates to Python to not interpret any backslashes in its usual way. We want regex to handle it!

example: re.search(r".+@.+\.edu", variable)
↳ just like f""

[] Set of characters [abcd] or [a-z] → a to z

[^] set of not allowed characters [^abcd] ex: r"[^@]+.edu → 1 or more; any char. except @

\w → Alphabetic/Numeric or _ (Built in, to be less cryptic)

→ There is much more!

(A|B) ex: (\w+|.)? [captured]

grouped! → once or twice or zero

(?:\w+|.) → Non captured

*Regular Expressions are quite cryptic; If you're solving smth familiar (like validating emails → libraries of the internet will be your friend - David Malan

resolve baby steps - David

Anything surrounded by () can be accessed through Var.groups() or .group(1) → 1st group

→ List (I think)

→ return value of re.search

.replace vs. re.sub() → same functionality, but allows reg. expr.s

22/3/2025

WEEK 8 - OOP

Procedural vs. functional vs. object oriented

tuple → datatype in Python: collection of values that are immutable

multiple

→ values cannot be changed (Raises TypeError if you try)

ex: return(name,house) → returns a tuple

, How to access? → name,house = fun() unpack values

→ student = fun() Tuple!

student[0], student[1]

(name) (house)

→ vs. return [name,house] square brackets

tuple vs. list → use a tuple when you don't values to be changed.

(Similar)

Creating your own data type

Classes & objects

keyword → class Student:

... → "ellipsis literal": Placeholder in Python indicating you'll write smth here
If you remove & leave empty: syntax Error

calls default constructor (not ideal practice)

student = Student()
student.name = input()

still can add attributes (↑//) to an empty class (doesn't standardize blueprint)

Constructor: → Has to come as 1st arg & by convention called "self"

def __init__(self, name, house):
    self.name = name
    self.house = house

Python's way for a constructor "initialize"

why? To access the current object! (like this in Java)

How to call? student = Student(name, house) ✓

Creates attributes & standardizes the class blueprint!

__ → "Dunder" Double underscore

Python handles memory management for us; so don't worry about these low-level details! (major advantage)
interpreter

Encapsulation: everything related to the class should be handled inside ex: data validation, get, set

---

raise ValueError("msg") → Good practice to raise an error, when data validation goes wrong

print(student) → prints memory location ✗ why? print() expects a string

How to override? __str__ "Dunder" method → Purpose: defines how object looks when printed!

def __str__(self):
    return "whatever"

→ Python Auto-Handles it with the object!

special methods in Python that you override & it just works; unlike your own unique class methods!

⚠ self has to be passed whenever creating a method, and as 1st argument

23/7/2025

Properties → An attribute that we have more control over! [why? unauthorized access, invalid format data, etc.]

Getters & setters in Python are defined through Properties. [no access modifiers like Public & Private]

"decorators": functions that modify other functions!

@property → syntax to define a property

Getter

```
def house(self):
    return self._house
```

@house.setter → syntax to define a property setter

```
def house(self, house):
    #data validation
    self._house = house
```

syntax to define a property

property name takes the attribute name [why? Error/Collision of names ⚠]

AND, attribute name begins with an _ [convention]

(In reality: attribute is now _ & property has the property name!)

when will the setter be called?
Anytime .house = as assignment operator happens
ex: in main() OR in __init__ (inside class)

when will the getter be called?
Anytime .house is accessed without an assign. operator

Advantage: data validation & all property related is in (1) place! (inside property)

⚠ Python relies on convention, not hard constraint.
You can still access the underlying _attribute: so by convention if you see an _ or __ attribute it's meant to be "private", so don't touch it, or it will be ur fault if things break. (even more emphasis)

---

int("18")
str(20)
str.lower()
list("hi")

In reality they're classes, & we're using their constructor & methods! → Most if not all things in Python are classes & objects!

How to determine data type, e.g., class? type(var) [one way of many] → once again "decorator"

Instance methods vs. class methods → How? @classmethod (doesn't have access

for each object | for the entire class | to "self", e.g., an instance/object)

Even further: class variables: exists as one copy for all instances

example:

```
class Hat:
    houses = ["Griff", "Slyth"]      → class variable
    @classmethod                     → decorator
    def sort(cls, name):             → "cls" instead of self
        # implementation             [convention]
        print(cls.houses)
```

References the class itself, since it's a classmethod now.

```
def main():
    Hat.sort("Harry")
```

No need to create an instance/object!

→ Access class variables or methods

→ How to call class constructor? cls(params), not Hat(params)

Good Practice: Have classes code into seperate files & import them.

Inheritance → class student(wizard): → Inherits from wizard

params

How to access Parent class? super(), & How to call it's constructor? super().__init__()

operator overloading → Achieved through dunder methods

```
def __add__(self, other):
    return class(other.//, //)
```

Inside main(): obj1 + obj2 (obj1 = self, obj2 = other)

26/7/2025

WEEK 9 - Etcetera (etc.)

- CS50 encourages Reading Documentation in general & utilizing Python's!

set → Auto-filters out duplicates ex: houses = set(), houses.add("1")
↳ not append

Global variables → can be Accessed, but can't be over-ridden (unless functions) like list()

↳ How to fix? At the beginning of each function write: global var

↳ why? variables are passed by value [a local copy, not original]

Read / write

ex:
balance = 0
def deposit(n):
    global balance
    balance += n

to functions!

- use global variables rarely; there are better solutions like OOP
↳ can get messy quickly

27/7/2025

- Python depends on the 'honor system': relies on convention & not constraints
↳ example: constraints like constants in python are calibrated

Dynamically typed vs. strongly typed language
↳ specifying datatypes ex: C++
↳ Auto-figures out data types → Python

pip install mypy
mypy program.py
↳ mistakes strongly typed languages, ex:
↳ no syntax error if run by: "python program.py"

def meow(n: int) → None:
    # Code
"Type Hint": expecting int
Returns None

Docstrings → placed 1st thing inside a function for documentation
"""
DOCumentation
"""
ex:
def meow(n):
    """ Meow n times """
↳ Documentation Convention that has support of many tools
ex: effort func() documentation to PDF

Pil install argclass Handles customization of cmd arguments e.g. --number

Unpacking → previously: first, last = input("name: ").split(" ")

→ *, forceful operator — def total(coin1, coin2):

has to match args length (or throw error)

#Code
coins = [100, 20] → or coin1, coin2 = [100, 20], total(c1, c2)
total(*coins) → similar to: total(coins[0], coins[1])

→ unpacking dictionaries, ** operator

dict = {"coin1": 50, "coin2": 100}
total(**dict)

Can a function have a variable number of arguments? Yes! ex: def fun(*args, **kwargs):

ex: print() utilizes them!

#Code
fun(10, 20, name="Ho")

A list containing positional arguments [10, 20]

A dict containing named arguments {"name": "Ho"}

def yell(*words):
    uppercased = [word.upper() for word in words]

// list(dict("Not written) Comprehensions"

Ethonic syntax to create a list "on the fly"

Even more functions: map/filter, enumerate!

Generators → functions that return massive amounts of data

Python keyword: yield (not similar to return), returns one value at a time

→ why? A big chunk of data 'returned' at once could crash the program.

ex: return "meow" * i vs. yield "meow" * i → can be 1 million times w/out crash

↳ still crash if too big

Thus your for loop is now considered an "iterator", iterates over the generated values one at a time.

CSSoc's final Project Brainstorm/notes - Attendance sheet exporter style

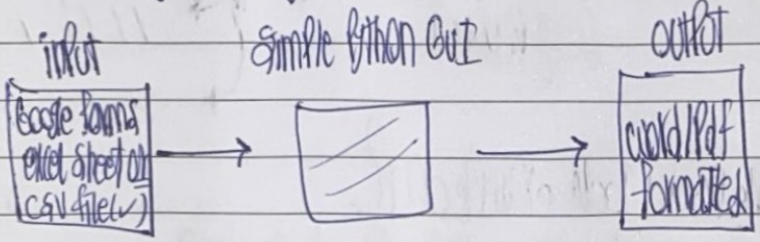
- first open google forms & try creating a simple example survey; and determine the type at which google exports the file: why? to determine whether it can be read using "csv" package or others.

Problem: Attendance gathering in HSA's technical sessions face the conflict of being time-consuming or complicated for no reason; whether through physical paper, writing inconsistent formats of names of each person, missing data, etc. (course code and soon)

Furthermore, the university requests a certain word/pdf format to accept the proposed attendees for the session; which becomes an overhead to manually fill out each field of data & becomes inefficient.

Solution:

Create a python program that automates this process. Through:



Even further; incase HSA decides to build an internal system for their club, and to avoid being a "traditional", simple python program, the program shall be runnable like ".exe" file with no need for python to be installed; furthermore, the functionality of the program should be package & import-friendly, to be officially installable through "Pip" package manager; meaning a solid system foundation should be thought of (like OOP, or functional). Lastly, utilize regular expressions to determine people/rows with missing data & act accordingly. Plus included tests!

*Generate tag before GUI *functions
*write tests!
Doc. revision! (end)
*Generate word Docs with current time naming!

tkinter vs. customtkinter
Built-in, really simple
↳ extension & more modern looking widgets

*Add title info for document
*Generate multi-datasets
1/2 ✓
1/2 X

Step-by-step Process:

*Make Email Row validation not required (if present in csv since it's not exported)

package to: Convert word to pdf, or vice-versa (more effic.)

① Test out Google forms & export CSV

*export time consistent format!

② Implement Backend

→ Read .csv file (csv module) & validate .csv extension (Regex) & (validators)
→ validate data fields using Regular expressions → Highlight invalid data-fields if any (missing or invalid format)
→ Generate word & pdf files (Python-docx & fpdf)
→ Before GUI: implement cmd arguments; when running (argparse)
- → python file.py file.csv --word
- → //// //// --pdf
- → ////// X (empty) Runs GUI (docx implemented)
→ write tests (pytest)

③ Customtkinter, GUI
→ AI-generated simple modern look example on PC.
→ validation for uis or change status msg!

④ Packaging
→ create ".exe" (pyinstaller)
→ pip installation & documentation process. (keep it simple!) → Harder, dont fr

*packages used → ReadMe → req.txt

Questions: How shall the system design look like → OOP? Procedural? Functional?
9/8/2023

Code from day one, with CS50's design constraints in mind to avoid refactoring

③ custom functions → shouldn't affect design, just treat as wrapper functions.

& project.py → later after almost done

@classmethod vs @staticmethod

→ use for stateless method (No instance needed, don't need anything inside class)

↳ use when there are class variables and a state of the class needs to be accessed through 'cls' instead of 'self'

(Not instance needed, But need smth inside class)