# CTD Python Essentials

Week1 – Introduction to Python

## Introducing Python

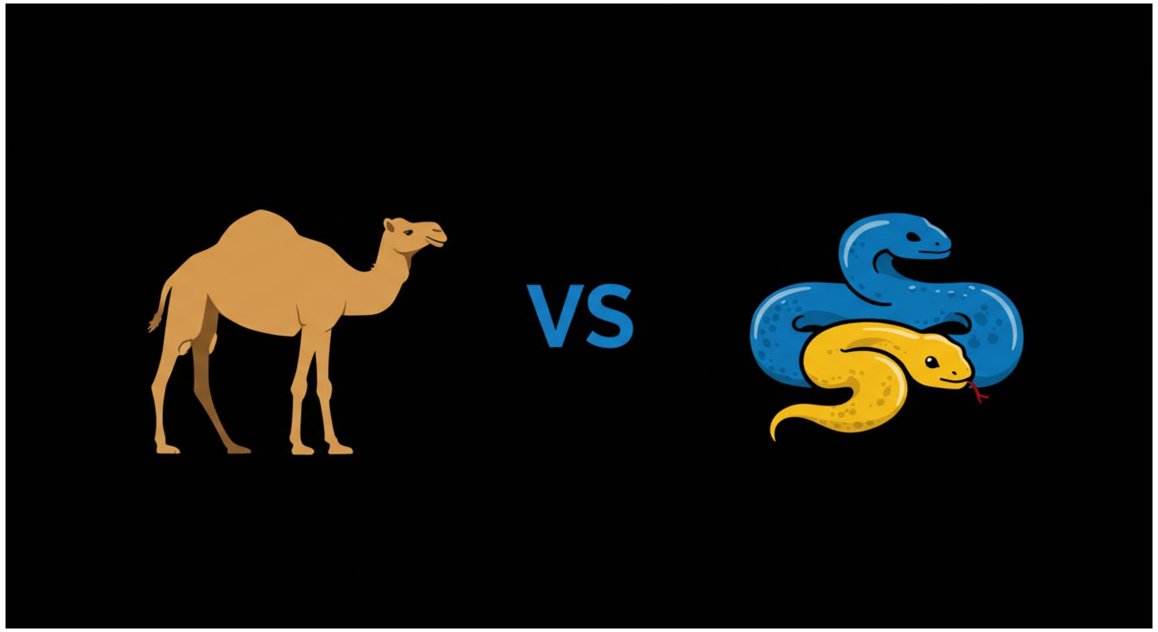
- Preeminent scripting and glue language
- Clean, simple syntax
  - “Executable pseudocode”
- Huge ecosystem
  - Rich standard library
  - Large number of high quality 3rd party packages
  - Vibrant community
- Object model

## Python Applications

- Important python domains (by no means comprehensive)
- numpy – Numerical Python
  - High performance mathematical operations
- scipy - Scientific Python
  - Advanced functions and algorithms built on numpy
- Machine learning
  - PyTorch, tensorflow, Keras, scikit learn and many more
- Web framework and APIs
  - Django, Flask
- Data Science and Engineering
  - Pandas for cleaning and manipulation
  - Matplotlib for visualization
  - Rpython for statistics
  - Sqlalchemy – database ORM
- Biopython for genomic data

## History

- Created in 1989 by Guido van Rossum at the Stichting Mathematisch Centrum in the Netherlands
  - A holiday hobby project
  - Based on previous work on a teaching language called ABC
- Evolved in the shadow of perl in the 1990s, and eventually superseded it as perl6 foundered.
- Perl: “There’s more than one way to do it.”
- Python: “There should be one—and preferably only one—obvious way to do it” – the pythonic way.
- Powerful binary libraries like numpy established python as the most important scripting language today
- The evolution of the language is governed by the PEP process. Check out:
  - Style guide: PEP8
  - Zen of Python: PEP20



## Python 2 vs 3

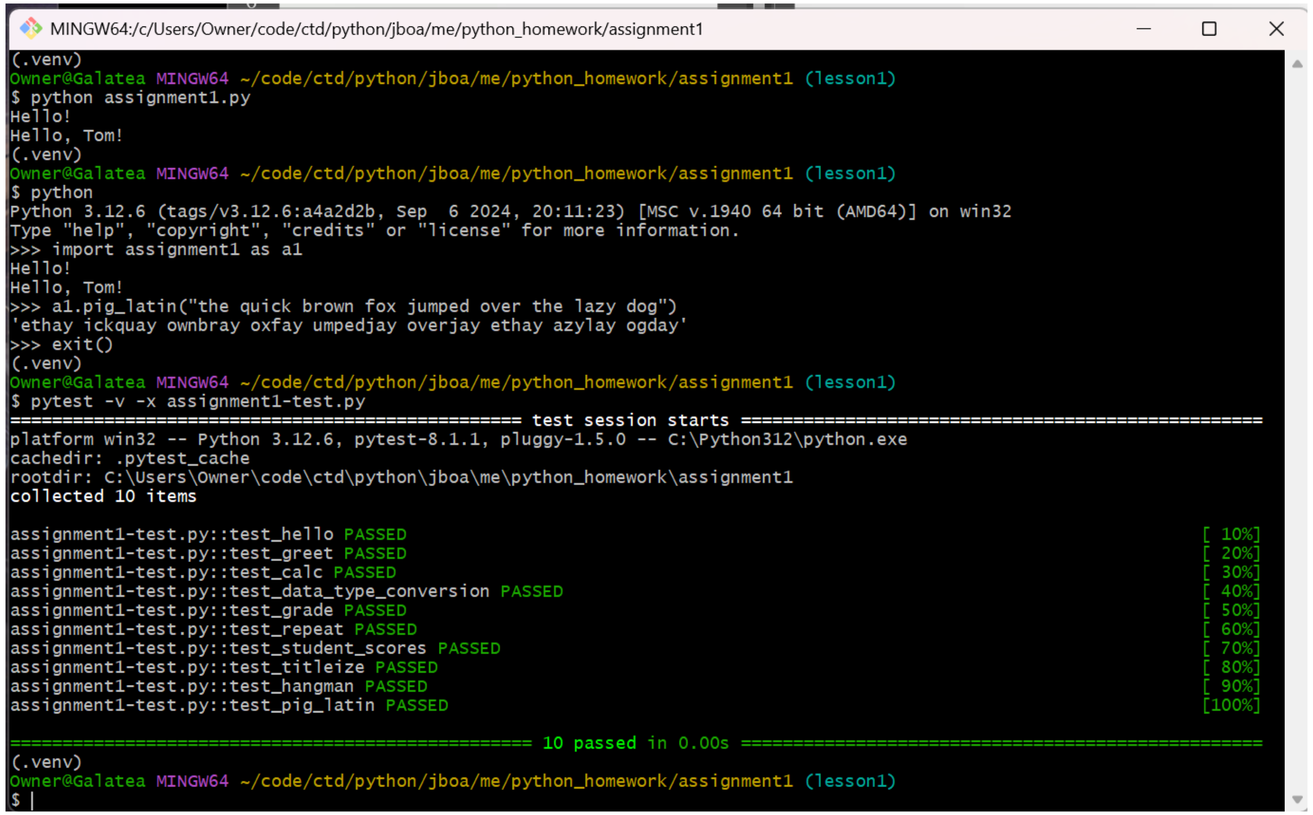
- Python 3 was introduced in 2008
    - Significant changes which are incompatible with python 2
    - Unicode support
    - Print as a function
    - Integer division
    - Range() returns an iterator
    - Exception syntax changes
- Python 2 was deprecated at the beginning of 2020 and should no longer be used
- Run python –version to check (two dashes)
- For backwards compatibility, some installations provide python as python3 and pip as pip3.

## Installing Python

- Lesson 1 provides detailed instruction on installing python and setting up the environment
  - CTD -> Python 100 v1 -> Introduction to Python
- There is extensive documentation on python installation and setup here:
  - Python Setup and Usage — Python 3.13.2 documentation
- The class uses several modules from pypi, the python package manager.
  - It is a convention to specify a list of requirements for a project in a file named requirements.txt.
  - Installed via `pip install -r requirements.txt`
- We will manage packages using a virtual environment (venv).
  - When the venv is created and activated, you should see (.venv) in your prompt.

# Running lesson examples and assignments

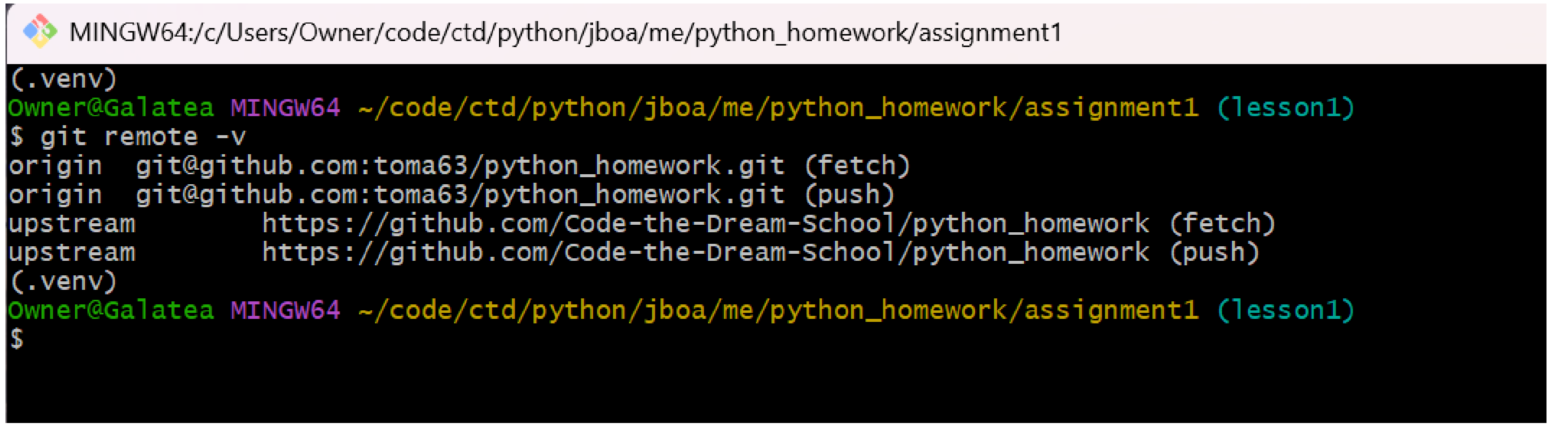
- The homework will be organized in a separate directory (folder) for each lesson.
  - The folder contains a file for you code and unit tests
  - e.g. assignment1.py and assignment1-test.py
- We recommend using a separate folder for running code samples from the lesson so the git workspace is not cluttered with temporary files.
- We are using Test Driven Development for this class
  - Write the tests first, then write code which passes the tests
  - Run tests at the bash (zsh on Mac) command line:
    - pytest -v -x assignment1-test.py
    - -v is optional – it lists passing tests
- Code samples and homework testing and debug can be run:
  - At the shell command line: python assignment1.py
  - In the python repl (Read Eval Print Loop):
    - >> Import assignment1 as a1
    - >> a1.my_func(my_args)
- Depending on your environment, you can exit the repl using ctl-c, ctl-d, or exit()
- If python hangs in git bash:
  - Add `alias python='winpty python.exe'alias python='winpty python.exe'` to ~/.bash_profile


MINGW64:/c/Users/Owner/code/ctd/python/jboa/me/python_homework/assignment1

```
(.venv)
Owner@Galatea MINGW64 ~/code/ctd/python/jboa/me/python_homework/assignment1 (lesson1)
$ python assignment1.py
Hello!
Hello, Tom!
(.venv)
Owner@Galatea MINGW64 ~/code/ctd/python/jboa/me/python_homework/assignment1 (lesson1)
$ python
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import assignment1 as a1
Hello!
Hello, Tom!
>>> a1.pig_latin("the quick brown fox jumped over the lazy dog")
'ethay ickquay ownbray oxfay umpedjay overjay ethay azylay ogday'
>>> exit()
(.venv)
Owner@Galatea MINGW64 ~/code/ctd/python/jboa/me/python_homework/assignment1 (lesson1)
$ pytest -v -x assignment1-test.py
================================================= test session starts ==================================================
platform win32 -- Python 3.12.6, pytest-8.1.1, pluggy-1.5.0 -- C:\Python312\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Owner\code\ctd\python\jboa\me\python_homework\assignment1
collected 10 items

assignment1-test.py::test_hello PASSED                                                                            [ 10%]
assignment1-test.py::test_greet PASSED                                                                            [ 20%]
assignment1-test.py::test_calc PASSED                                                                             [ 30%]
assignment1-test.py::test_data_type_conversion PASSED                                                             [ 40%]
assignment1-test.py::test_grade PASSED                                                                            [ 50%]
assignment1-test.py::test_repeat PASSED                                                                           [ 60%]
assignment1-test.py::test_student_scores PASSED                                                                   [ 70%]
assignment1-test.py::test_titleize PASSED                                                                         [ 80%]
assignment1-test.py::test_hangman PASSED                                                                          [ 90%]
assignment1-test.py::test_pig_latin PASSED                                                                        [100%]

================================================== 10 passed in 0.00s ==================================================
(.venv)
Owner@Galatea MINGW64 ~/code/ctd/python/jboa/me/python_homework/assignment1 (lesson1)
$ |
```

## Homework repository setup

- Detailed instructions are provided in the homework repo README.md file.
- Rather than forking, you will:
  - Create a new empty repo called python_homework
  - Clone the school's python_homework repo
  - Change the remote to your repo
  - Set the school's repo as the upstream
- This makes the default base for pull requests your repo, rather than the school's
- We may occasionally ask you to get the latest changes from the school's repo:
  - git fetch upstream
  - git checkout main
  - git merge upstream/main



```
MINGW64:/c/Users/Owner/code/ctd/python/jboa/me/python_homework/assignment1
(.venv)
Owner@Galatea MINGW64 ~/code/ctd/python/jboa/me/python_homework/assignment1 (lesson1)
$ git remote -v
origin  git@github.com:toma63/python_homework.git (fetch)
origin  git@github.com:toma63/python_homework.git (push)
upstream        https://github.com/Code-the-Dream-School/python_homework (fetch)
upstream        https://github.com/Code-the-Dream-School/python_homework (push)
(.venv)
Owner@Galatea MINGW64 ~/code/ctd/python/jboa/me/python_homework/assignment1 (lesson1)
$
```

## Syntax and scoping

- Python has a sparse, clean syntax
  - No {} to specify blocks
  - No ; to end lines/statements
- It is unusual in that it uses indentation to specify block structure
- Lines can be continued using a '\' character
- Open (), {}, and [] also allow line continuation
- Multiline strings use """
- Comments start with # and go to the end of the line
  - You can also use """ for multiline comments such as docstrings
- Python does not have block (lexical) scoping.  It uses function scope.

## Variables, datatypes, conversion

- Variables are created by assigning to them
  - There is usually no declaration although they can be declared explicity global
- The convention is to use lowercase names separated by '_' (snake case, of course)
- Basic datatypes
  - int, float, bool, complex
- Sequential datatypes
  - list, tuple, str
- Mapping (associative array): dict
- Set: set
- None: explicit lack of a value
- Explicit conversion (recommended) use the name of the datatype:
  - int(), float(), bool(), complex(), list(), tuple(), str(), set(), dict()

## Strings

- A sequence of Unicode characters
- Python provides a rich set of string methods
- Strings are immutable
- All of the immutable sequence methods can be used for strings

# Truth in Python

- The following are False in python
  - False
  - None
  - Zero of any numeric type
  - Empty sequences and collections: (), [], {}, set()
- All other values are considered True

# Operators

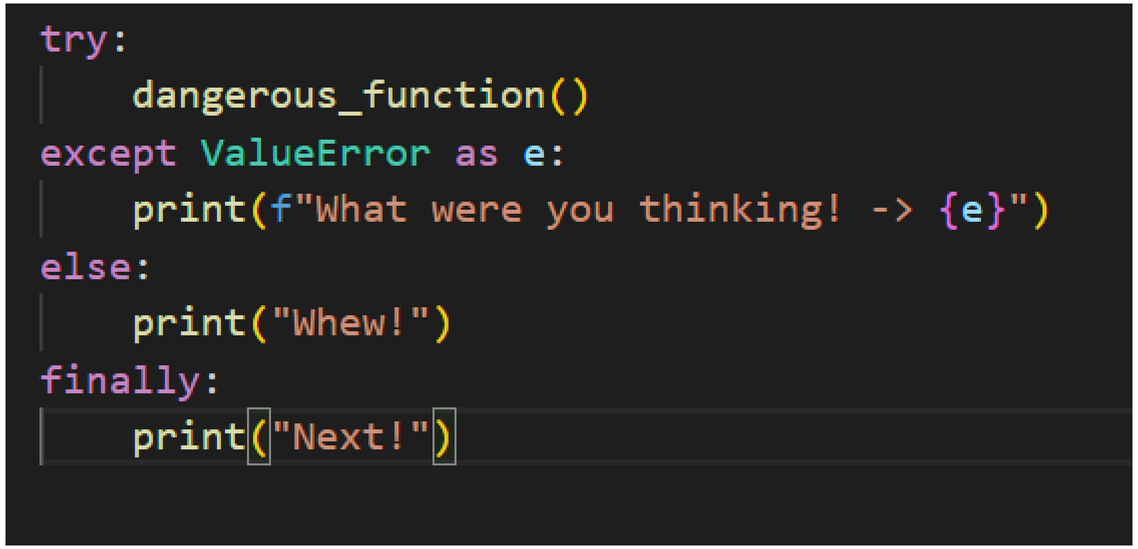
- Arithmetic:
  - + (addition): 3 + 2 → 5
  - - (subtraction): 5 - 3 → 2
  - * (multiplication): 4 * 2 → 8
  - / (division): 9 / 3 → 3.0
  - // (integer division): 9 // 3 → 3
  - % (modulus, remainder): 7 % 3 → 1
  - ** (exponentiation): 2 ** 3 → 8
- Comparison:
  - == (equal to): 5 == 5 → True
  - != (not equal to): 5 != 4 → True
  - < (less than): 3 < 4 → True
  - > (greater than): 10 > 5 → True
  - <= (less than or equal to): 5 <= 5 → True
  - >= (greater than or equal to): 7 >= 3 → True
- Logical:
  - and: True and False → False
  - or: True or False → True
  - not: not True → False
  - Not to be confused with bitwise operators
- Bitwise:
  - & (bitwise and): 0xF0 & 0x0F → 0x00
  - | (bitwise or): 0xF0 & 0x0F → 0xFF
  - ^ (bitwise exclusive or): 5 ^ 3 → 6
  - ~ (bitwise not): ~0b0101 → -6 (2's complement)
  - << (left shift): 5 << 1 → 10
  - >> (right shift): 0b1010 >> 1 → 5 (0b0101)

# Control flow

- A test is anything which returns a True or False value
    - Or can be coerced to True/False
- Assignment does not return a value in Python
    - So: while foo = makeSomething(): # doesn't do what you might think
- if <test>: [elif <test>:] [else:]
- while <test>:
- for <iteration-variable> in <sequence>|<iterator>:
    - Python does not have the c style for (;;) loop
    - range(<stop>) returns an iterator (starting with 0) which is often used in for loops
    - Range also supports range(<start>, <stop>[, <step>])
- break can be used to jump out of a loop
- continue can be used to skip to the next iteration

## Exceptions

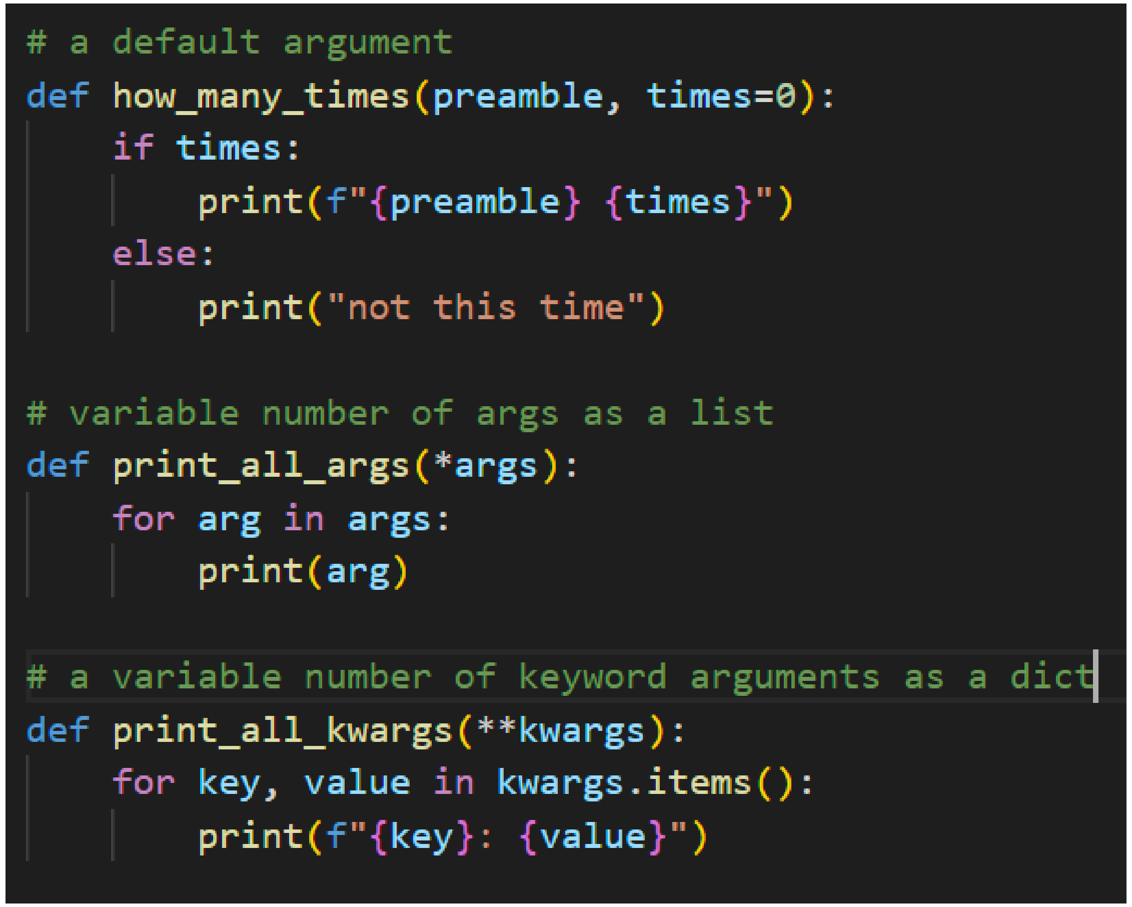
- Python provides Exceptions to trap and manage errors and other exceptional conditions.
- try:
  - Start of block of code which will be trap exceptions
- There can be one of more except: blocks to capture specific exceptions.
- else: clauses can be added which execute of an exception is not found
- A finally: block can be added which runs regardless of exceptions
- Exceptions can be nested
- Custom exceptions can be created
- Exceptions aren't exceptional
  - A well design program should anticipate things which can go wrong



```python
try:
    dangerous_function()
except ValueError as e:
    print(f"What were you thinking! -> {e}")
else:
    print("Whew!")
finally:
    print("Next!")
```

# Functions

- Functions are created using the def keyword
- A function establishes an enclosing variable scope
- Functions can be nested
- Python has first class functions, they can be assigned to variables
- Arguments are passed by reference
- Default arguments are supported (e.g. numerical_arg=0)
- The entire argument list can be referenced as a list (*args)
- The entire argument list can be referenced as a keyword dict: (**kwargs)
- Python supports a simple type of anonymous function called a lambda
    - lambda arg[..., arg]: <single expression>



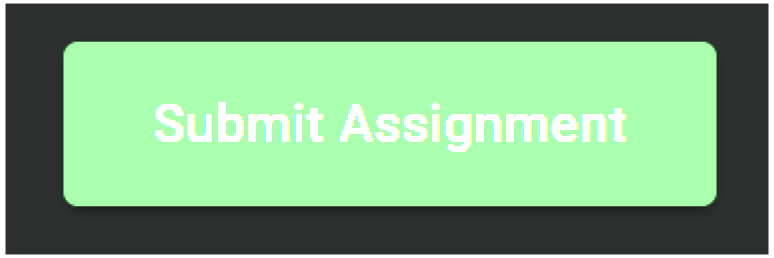
```python
# a default argument
def how_many_times(preamble, times=0):
    if times:
        print(f"{preamble} {times}")
    else:
        print("not this time")

# variable number of args as a list
def print_all_args(*args):
    for arg in args:
        print(arg)

# a variable number of keyword arguments as a dict
def print_all_kwargs(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")
```

# Debugging

- Unit tests
  - Checks correctness explicitly and can provide details on what's wrong
- Use print() – add as needed to see what's going on
- The traceback module used in conjunction with exceptions
  - Example in lesson 2
  - Shows where the failure is in the function call stack
- The logging module
  - Varying levels and control of verbosity
- The python debugger in conjunction with a vscode plugin – lesson video

# Submitting assignments

- Make sure unit tests are passing
- As with other classes:
  - Create a branch for all the code and other files in the assignment
  - Commit all your changes to the branch
  - Push the branch to your python_homework repo
  - Use the branch to create a pull request
- Submit your lesson including the pull request:
- Answer the other questions in the form



## Demo and Q&A