Writing a logger in 30 minutes

Scott Little KI5WLJ

About Me

My name is Scott (KI5WLJ). I was licensed a year ago, and have been programming for almost six years in languages such as Python 3, JavaScript, Rust and Flutter.

I’ve competed in about 7 programming contests and won two awards.

With amateur radio, my father, my brother, and I go out to do Parks on the Air frequently.

Goals

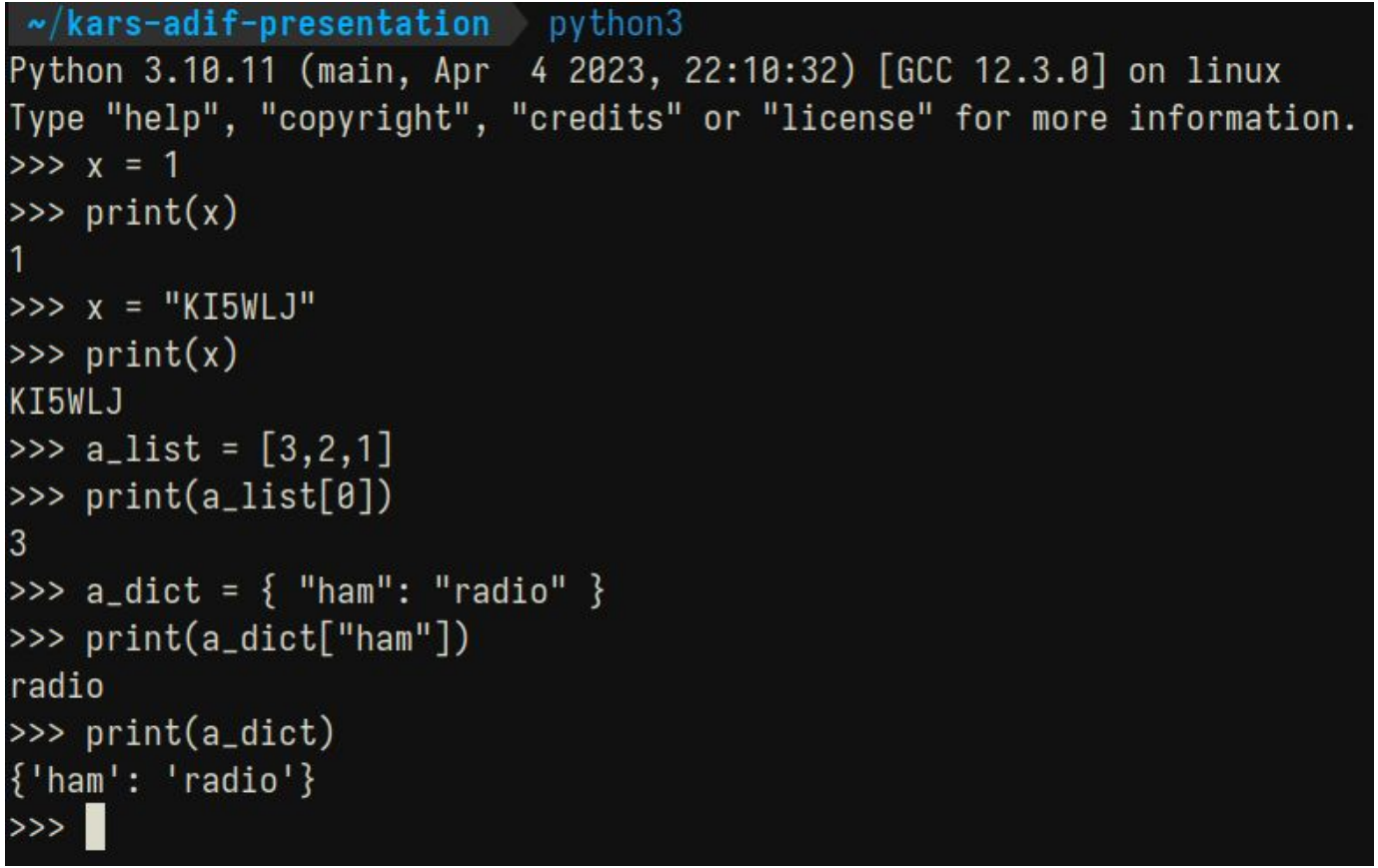
- To understand basic Python programming
- To understand the basic structure of ADIF *tags*, *records*, and *files*.
- To write a logger that can successfully upload a QSO to the ARRL Logbook of the World

Python 101

Variables

A variable is a *name* that has a *value*. If you know what a variable is in mathematics, you know what a variable is in Python.

Variables have only one operation on them in Python: assignment. Assignment has two parts: naming the variable and giving it a value.



Basic variable types

Functions and classes

A function is a named block of code that you can reuse and may return a value. (`f(x) = x`^{2} is a function)

print() is a function, just like `input()`

Functions are written like this:

```
1 def a_function(argument1, arg2, ...):
2     x = 0
3     another_function()
4     return x
```

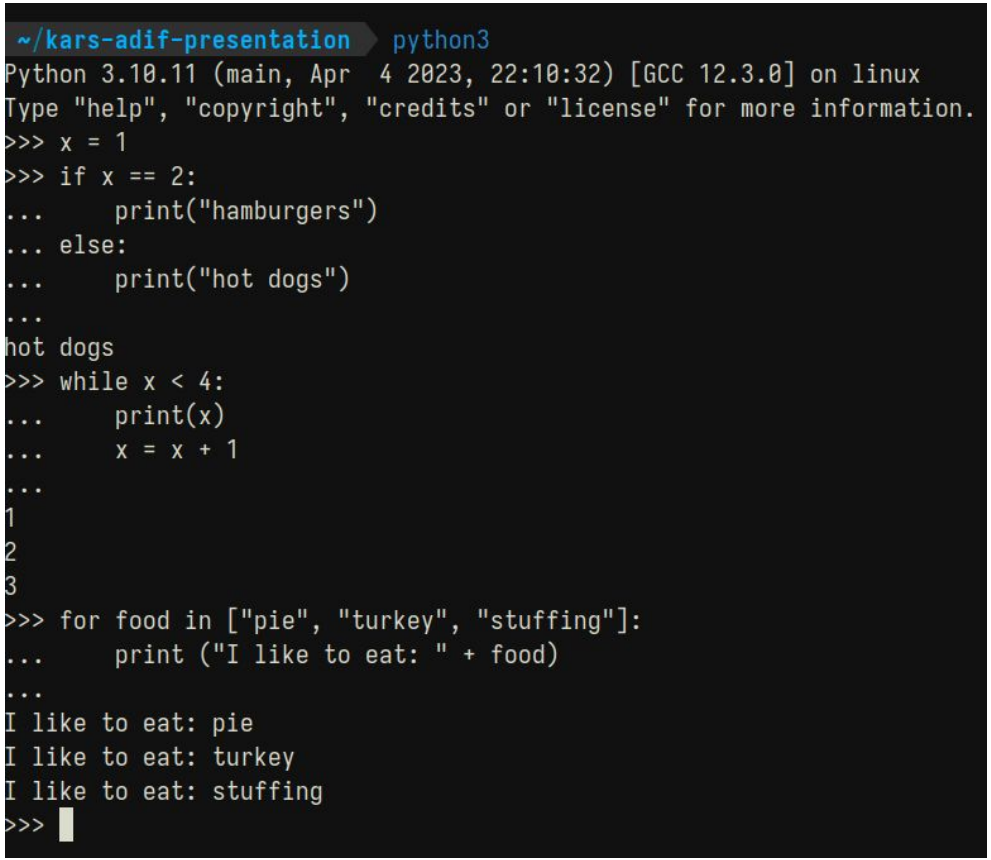
Functions can use any type of code.


~ python3
Python 3.10.10 (main, Feb 7 2023, 12:19:31) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> def f():
... print("de KI5WLJ")
...
>>> f()
de KI5WLJ
>>> def add(a, b):
... return a + b
...
>>> add(2, 2)
4
>>> add(2, 2) + 2
6
>>>

Control flow

`if` is used to conditionally execute a block of code.

`for & while` loops are used to conditionally repeat a block of code.



If - Elif - Else, For loops, and While loops

Comparison Operations

Equality: `A == B`

Inequality: `A != B`

Ordering:

Less than: `A < B`, Greater than: `A > B`

Less than or Equal to: `A <= B`, Greater than or Equal to: `A >= B`

“=” vs “==”

Programmers pronounce “x = 1” as “X has the value of one”, or “X is one”.

In Python, “=” DOES NOT MEAN EQUALS.

To say “Does X equal one?”, a programmer would write “X == 1”

Number operations

Addition, denoted by a +

Subtraction, denoted by a -

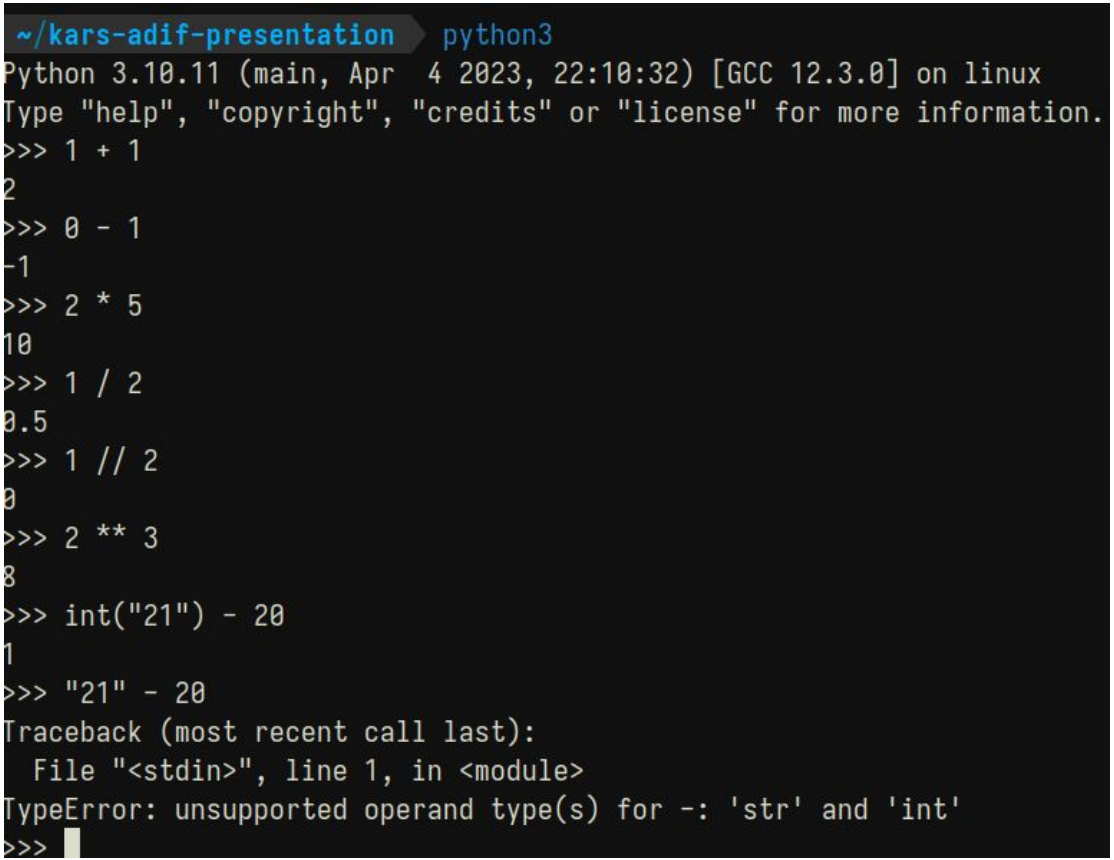
Multiplication, denoted by a *

Division, denoted by a /

Floor division, denoted by // (rounds to lowest whole number)

Exponentiation, denoted by **

Casting, converting a string to a number, denoted by `int(<something>)`



Number operations

String operations

Indexing: Read a single character: `“KI5WLJ”[0] == “K”(0 is the first element, 1 is 2nd)`

Slicing: Reading only a part of a string: `“KI5WLJ”[1:4] == “I5W” (does not include the final char)`

Replacing: `“KI5WLJ”.replace(“5”, “4”) == “KI4WLJ”`

Splitting: `“KI5WLJ”.split(“5”) == [“KI”, “WLJ”]`

Searching: Find the index of another string: `“KI5WLJ”.index(“5”) == 2`

Length: `len(“KI5WLJ”) == 6`

Casting: turning a number into a string: `str(1) + “1” == “11”`

```
~/kars-adif-presentation python3
Python 3.10.11 (main, Apr  4 2023, 22:10:32) [GCC 12.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> s = "I am a person"
>>> s[0]
'I'
>>> s[4]
' '
>>> s[5:13]
'a person'
>>> s[5:]
'a person'
>>> s[:5]
'I am '
>>> s.replace("person", "ham")
'I am a ham'
>>> s
'I am a person'
```

```
>>> s = s.replace("person", "ham")
>>> s.split(" ")[3]
'ham'
>>> s.split(" ")
['I', 'am', 'a', 'ham']
>>> s.index("a")
2
>>> s[2]
'a'
>>> len(s)
10
>>> str("1") + "2"
'12'
>>>
```

String operations

List Operations

Indexing: get the value at a particular index: `[3, 2, 1][1] == 2`

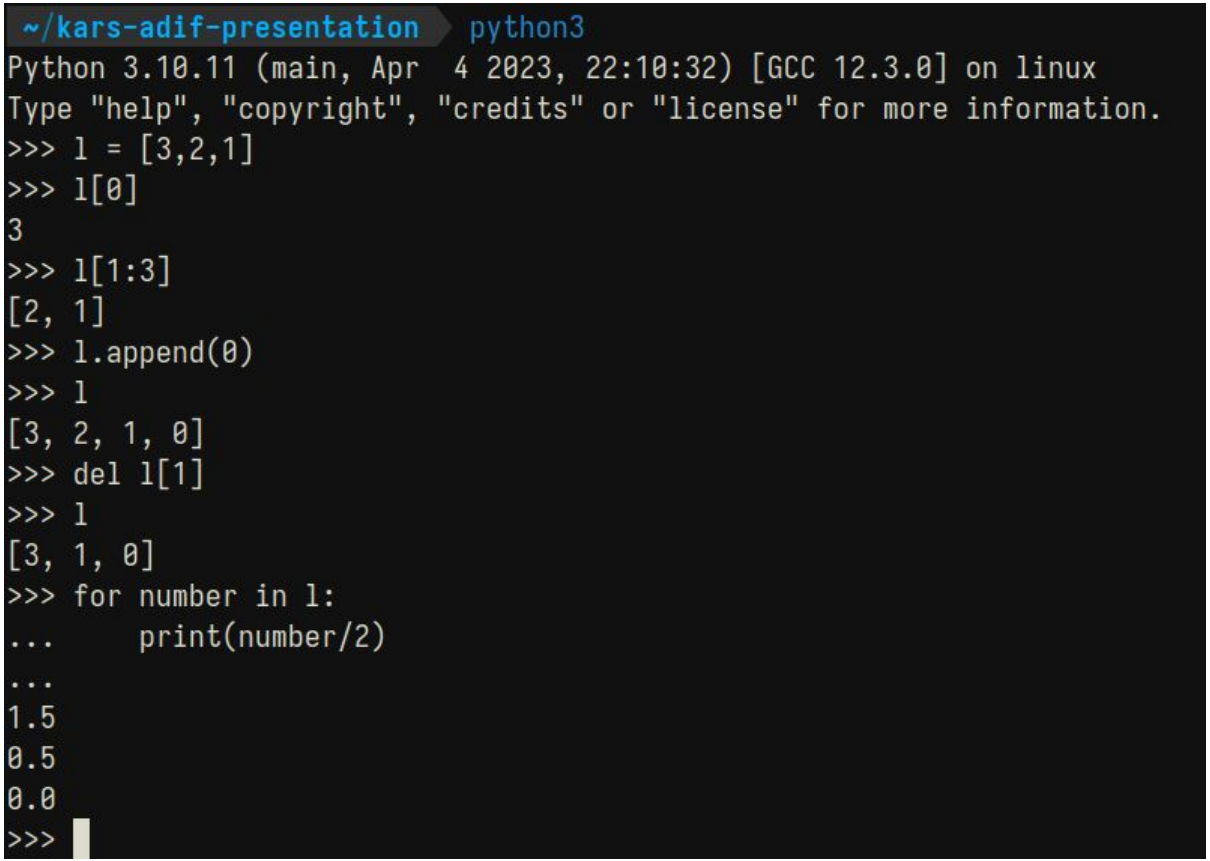
Slicing: getting a part (*slice*) of the list: `[3, 2, 1][0:2] == [3, 2]`

Appending: adding a value to the end of a list: `[1, 2].append(3) == [1, 2, 3]`

Deleting: removing a value from the list: `x = [1, 2, 3]; del x[0]; x == [2, 3]`

Iteration: doing something to each value: the below example calls `process` on each element of list.

```
for x in list:
process(x)
```



List operations

Dictionary Operations

Dictionaries have *keys* that are mapped to *values*

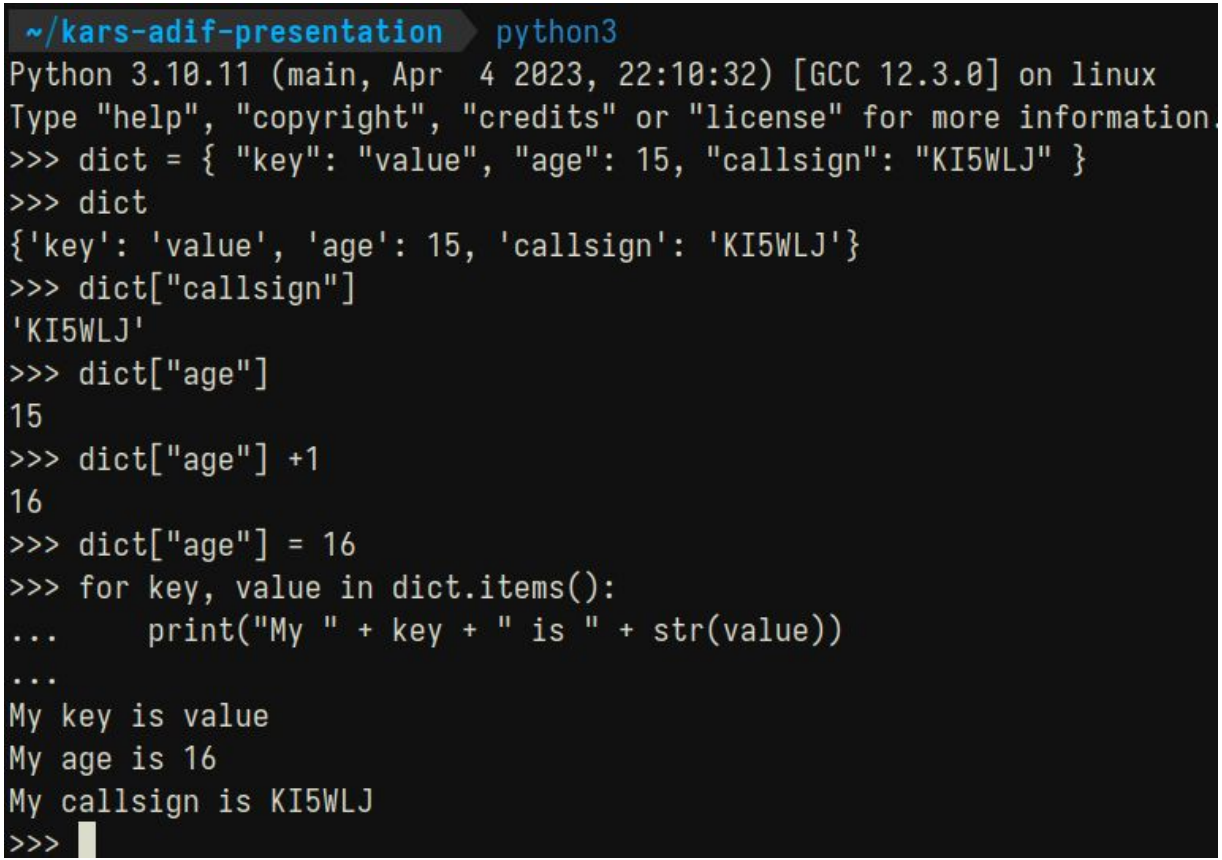
Dictionaries are defined like this: `{ “key”: “value”, “key2”: 3.14 }`.

Usually, keys are strings, while values can be numbers, strings, lists or even other dictionaries.

You can *access* an array: `dict[“key2”]` to get the value (`3.14`).

You can *assign* to an array: `dict[“key2”] = 6.28` to set the value

To iterate over a dictionary, use `for key, value in dict.items()`



Dictionary Operations

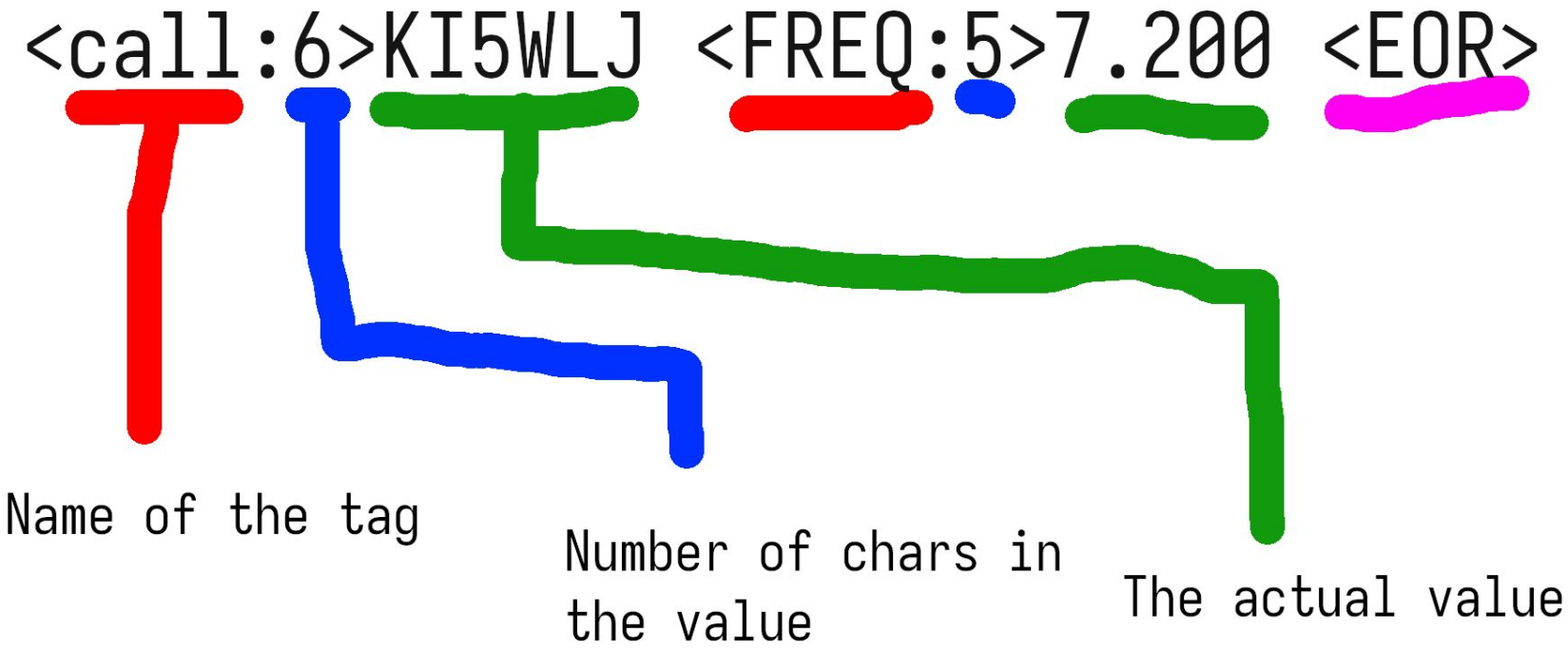
ADIF Basics

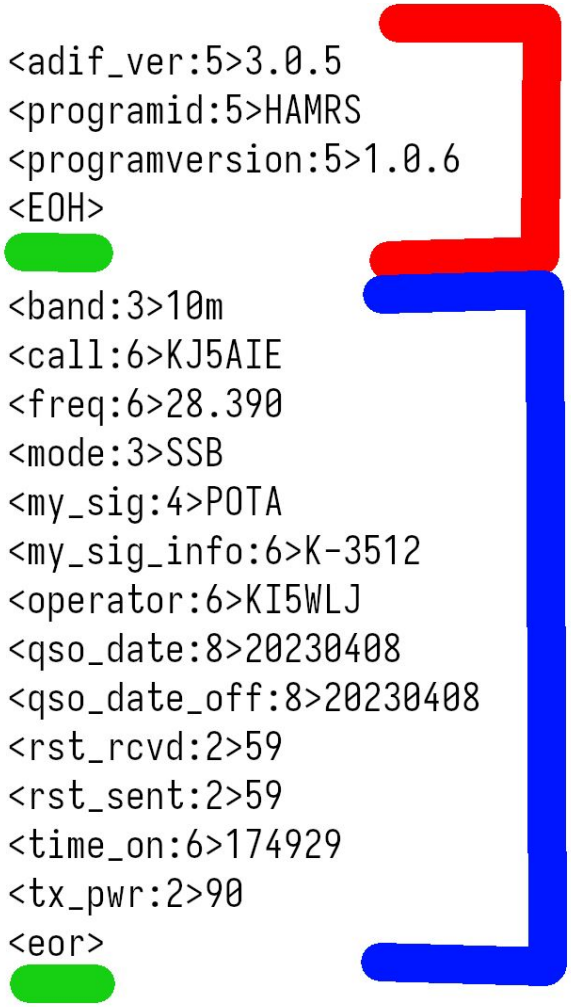
ADIF (https://adif.org)

ADIF is a common format to exchange log data. It takes the format of:

- One or more *tags* in the header
- The text <eoh>
- One or more records, where:
  - Each record contains one or more tags, and
  - The text <eor>

What is a tag?


<call:6>KI5WLJ <FREQ:5>7.200 <EOR>
Name of the tag
Number of chars in the value
The actual value



Structure of an ADIF file:

The header has a few special tags to help programs read it. <eoh> ends the header.

This file contains one record. Like the header, it has many tags.

Each record is ended with an <eor> (end of record).

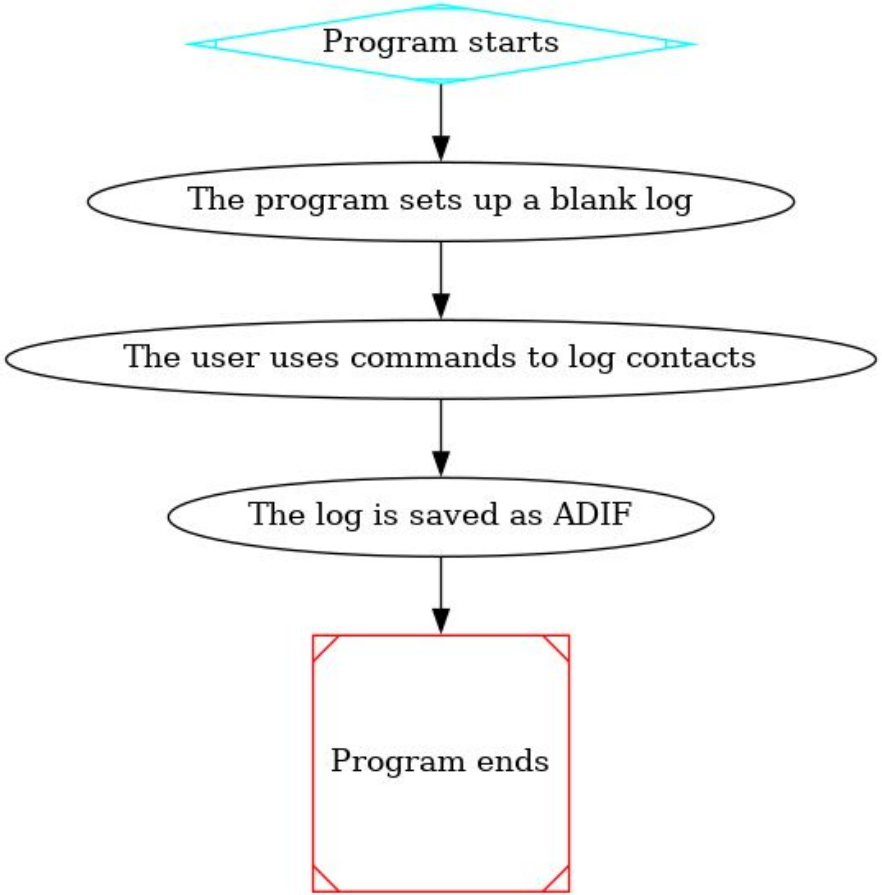
Required fields

The ARRL's *Logbook of the World* requires a few tags to have a complete QSO:

- CALL (The other station's callsign)
- FREQ (Your frequency in MHz)
- MODE (One of a few defined modes, but we will only handle SSB)
- QSO_DATE (The date the QSO started)
- TIME_ON (The time the QSO started)
- STATION_CALLSIGN (The callsign you used on the air)

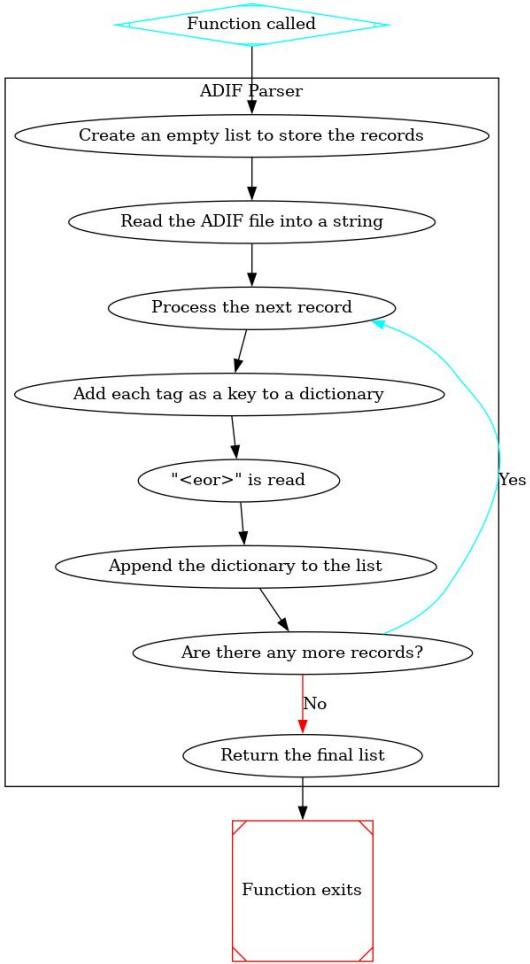
Writing the logger

Overview


Program starts
The program sets up a blank log
The user uses commands to log contacts
The log is saved as ADIF
Program ends

The ADIF Parser

To parse ADIF, we have to discard the header, then read the tag name, tag length and then read *length* number of chars after the end of the tag data.



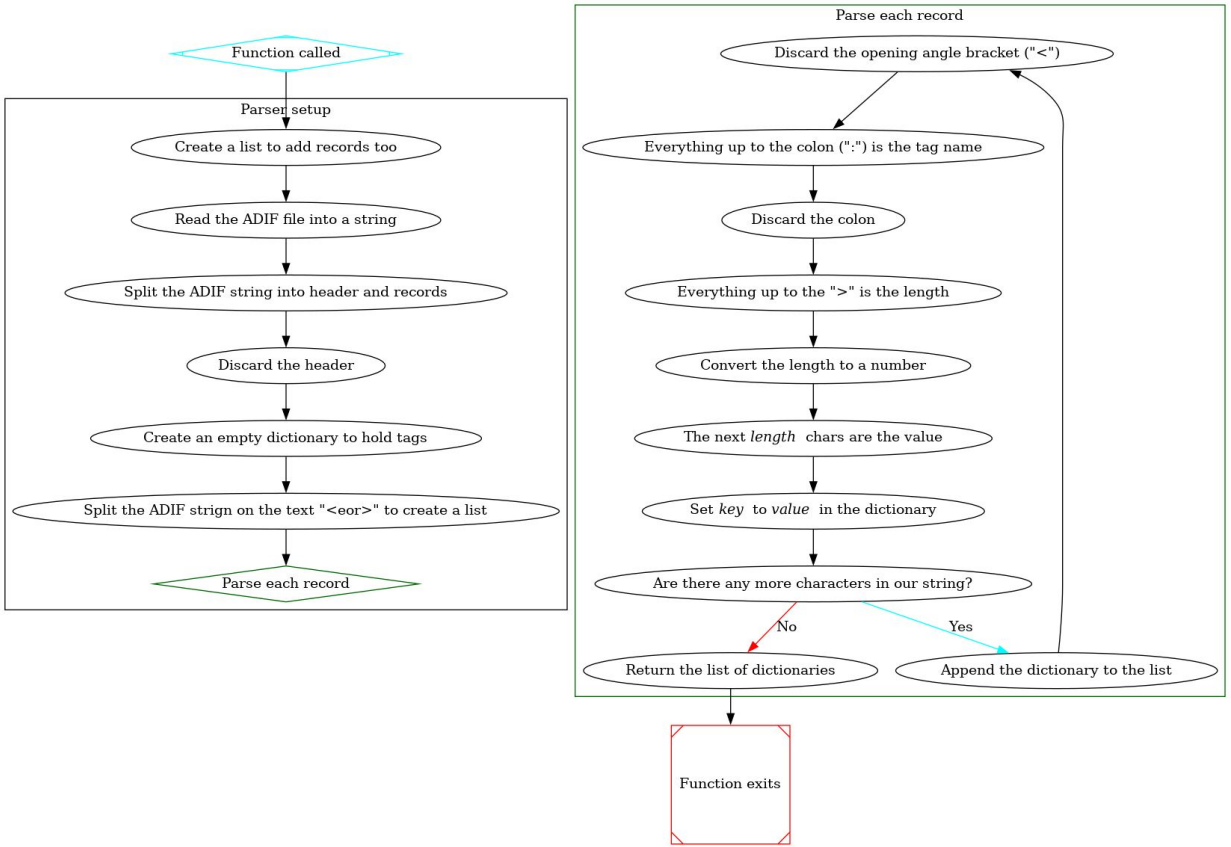
```python
def parse_adif(filename):
    output = [] #Step 1
    with open(filename) as adif: # Step 2
        data = adif.read()

    header_and_records = data.split("<EOH>") # Step 3
    del header_and_records[0]
    records = header_and_records[0]
    records = records.split("<eor>")

    for record in records: # Step 4
        record_dict = {}
        record = record.strip().replace("\r","").replace("\n","") # Remove whitespace at the start and end
        while len(record.strip()) > 0: # Do this until there is no more data to be read
            record = record[1:] # Skip the first <
            print(record)
            colon_loc = record.index(":")
            tag_name = record[:colon_loc]
            record = record[colon_loc+1:] # Skip the colon by adding one

            angle_loc = record.index(">")
            length = record[:angle_loc]
            length = int(length) # Convert length to a number
            record = record[angle_loc+1:] # Skip the angle bracket

            value = record[:length]
            record = record[length:]
            record_dict[tag_name] = value # Step 5

        if record_dict != {}: # Step 6
            output.append(record_dict);
    return output # Step 7
```

Create a list and read the file

Remove the header

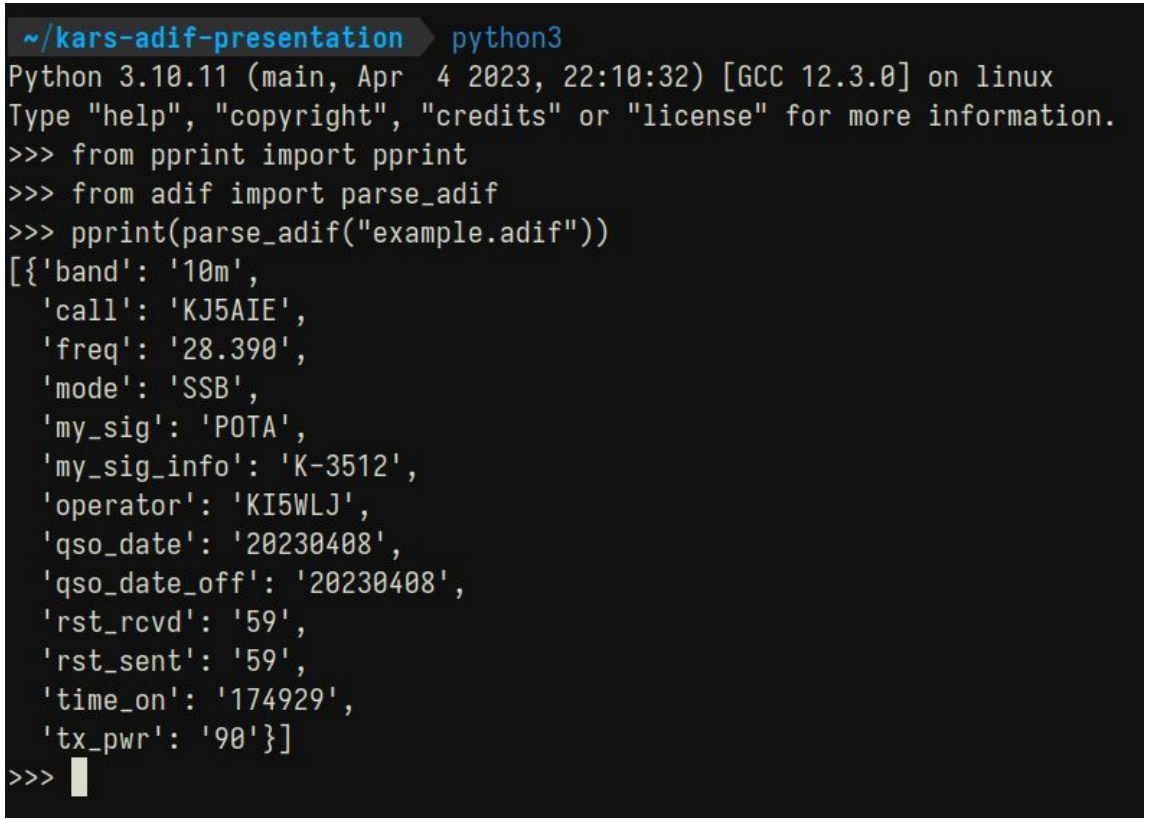
Add each tag to a dict

Add the dict to a list and return it



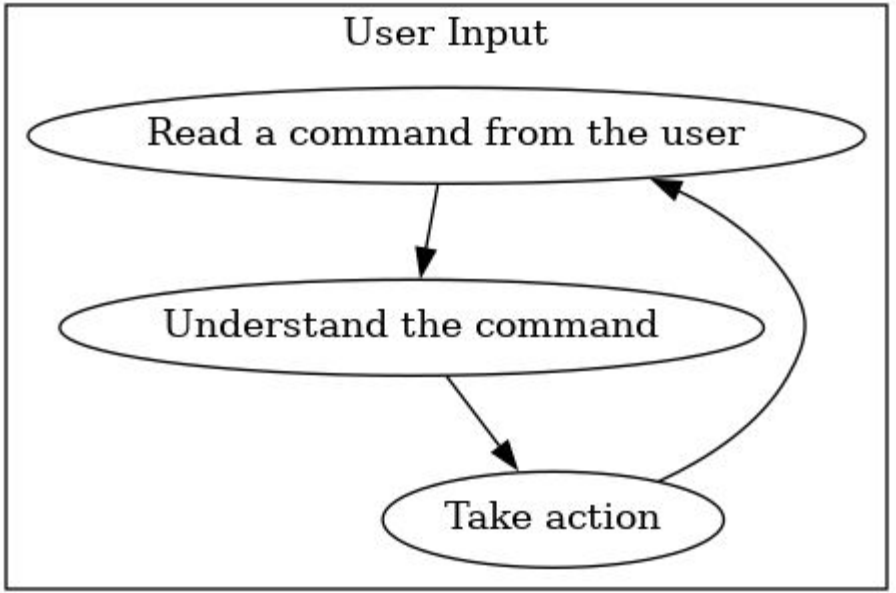
Full diagram of the parser

Testing the parser



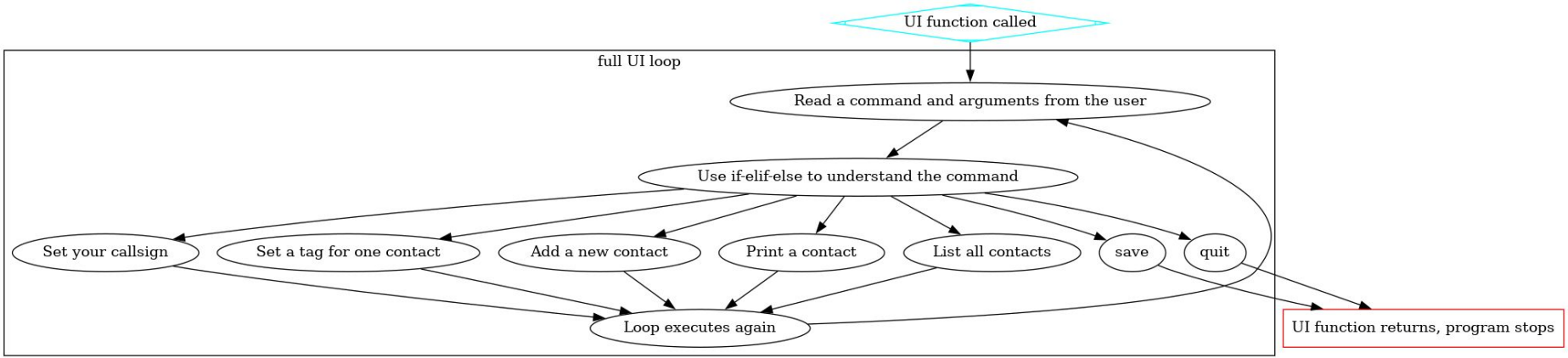
The User Interface

We will be making a command-line interface, where users type commands such as “cd /” or “chkdisk”



```
45 def ui(adif_data):
46     from pprint import pprint # Add an included function to print data in a more attactive way
47     import datetime # We use this to get the time of day
48     callsign = ""
49     while True:
50         command = input("enter command> ")
51         x = command.split(" ")
52         action = x[0]
53         arguments = x[1:]
```

The main UI loop



How all the commands fit together

Quit, List, and Print

To quit, we use the keyword `break`, which exits the loop.

To list, we iterate over each contact and print the callsign, a tab, then the frequency

To print, we get an index from the user and print out all of that contact's data.

```
if action == "quit":
    break # Exit the loop
elif action == "list":
    for record in adif_data:
        print(record["call"] + "\t" + record["freq"])
elif action == "print":
    idx = int(arguments[0])
    pprint(adif_data[idx])
```

Call & Set

We use a simple assignment to set the user's callsign.

For set, we use "double-indexing", because adif_data[idx] returns a dictionary which we then index to assign the value to it.

```
elif action == "call":
    callsign = arguments[0]

elif action == "set":
    idx = int(arguments[0])
    target = arguments[1]
    value = arguments[2]
    adif_data[idx][target] = value
    print(target + " of record " + str(idx) + " set to: " + value)
```

Adding a contact

We get the callsign and frequency from the user, then use the datetime module that's included with Python to get the current UTC time. We then use some included functions to get the formats YYYYMMDD and HHMM for ADIF

```
67        elif action == "add":
68            call = input("callsign> ")
69            freq = input("freqency> ")
70
71            now = datetime.datetime.now(datetime.timezone.utc)
72            qso_date = now.strftime("%Y%m%d") # Get a string in the format YYYYMMDD
73            time_on = now.strftime("%H%M")
74
75            record = {
76                "call": call,
77                "freq": freq,
78                "mode": "SSB",
79                "qso_date": qso_date,
80                "time_on": time_on,
81                "station_callsign": callsign
82            }
83            adif_data.append(record)
```

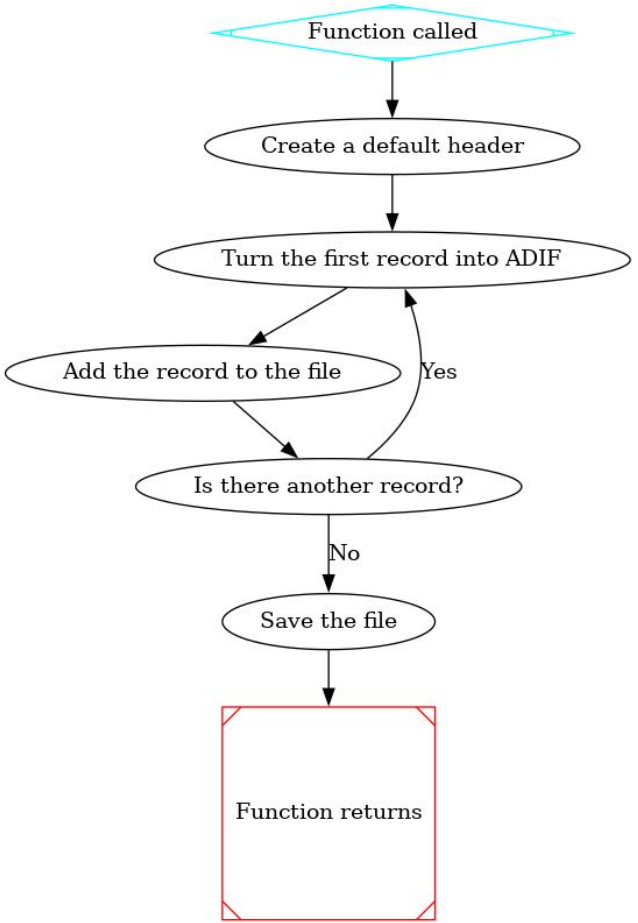
Saving contacts

Here, we simply take the filename from the user and pass it to the `save()` function we are about to write.

```
86          elif action == "save":
87              filename = arguments[0]
88              save(adif_data, filename)
89              break
```

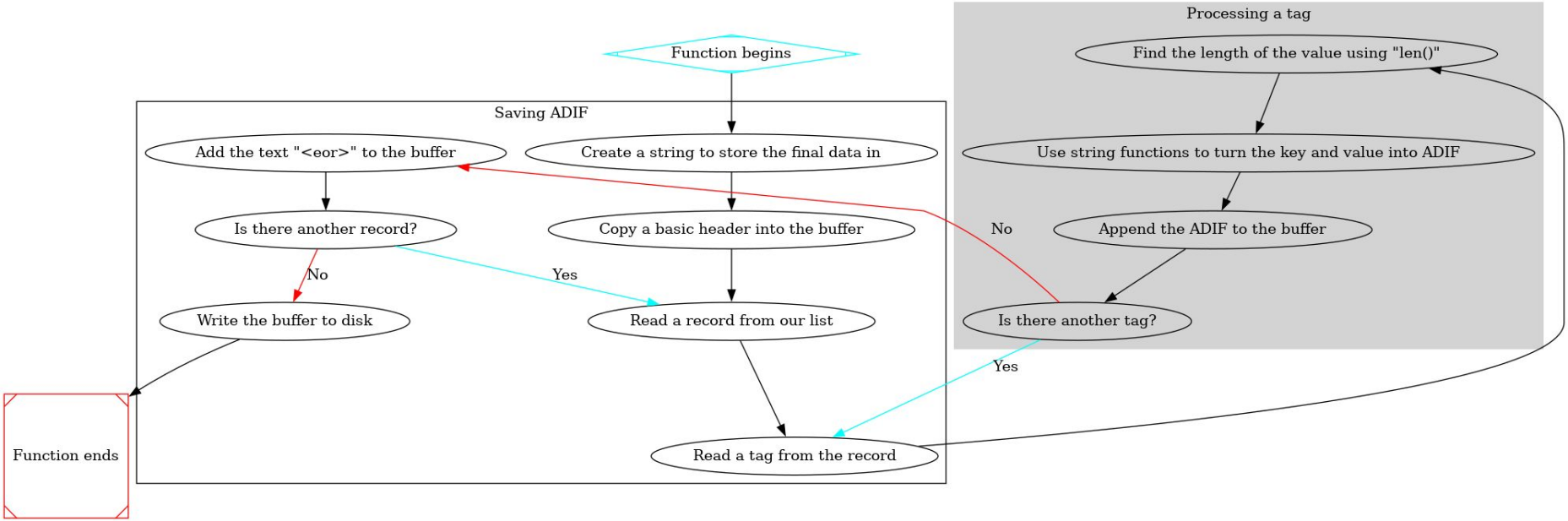
Saving the log

This is the easiest part — only ten lines of code.



```
33 def save(adif_data, filename):
34     buffer = "";
35     buffer = buffer + "<adif_ver:5>3.1.4<programid:4>KARS<EOH>\n"
36     for record in adif_data:
37         for key, value in record.items():
38             length = str(len(value)) # Convert the length to characters rather than a number
39             adif = "<"+key+":"+length+">"+value
40             buffer = buffer + adif
41         buffer += "<eor>\n"
42     with open(filename, "w") as f:
43         f.write(buffer)
```

Our save function



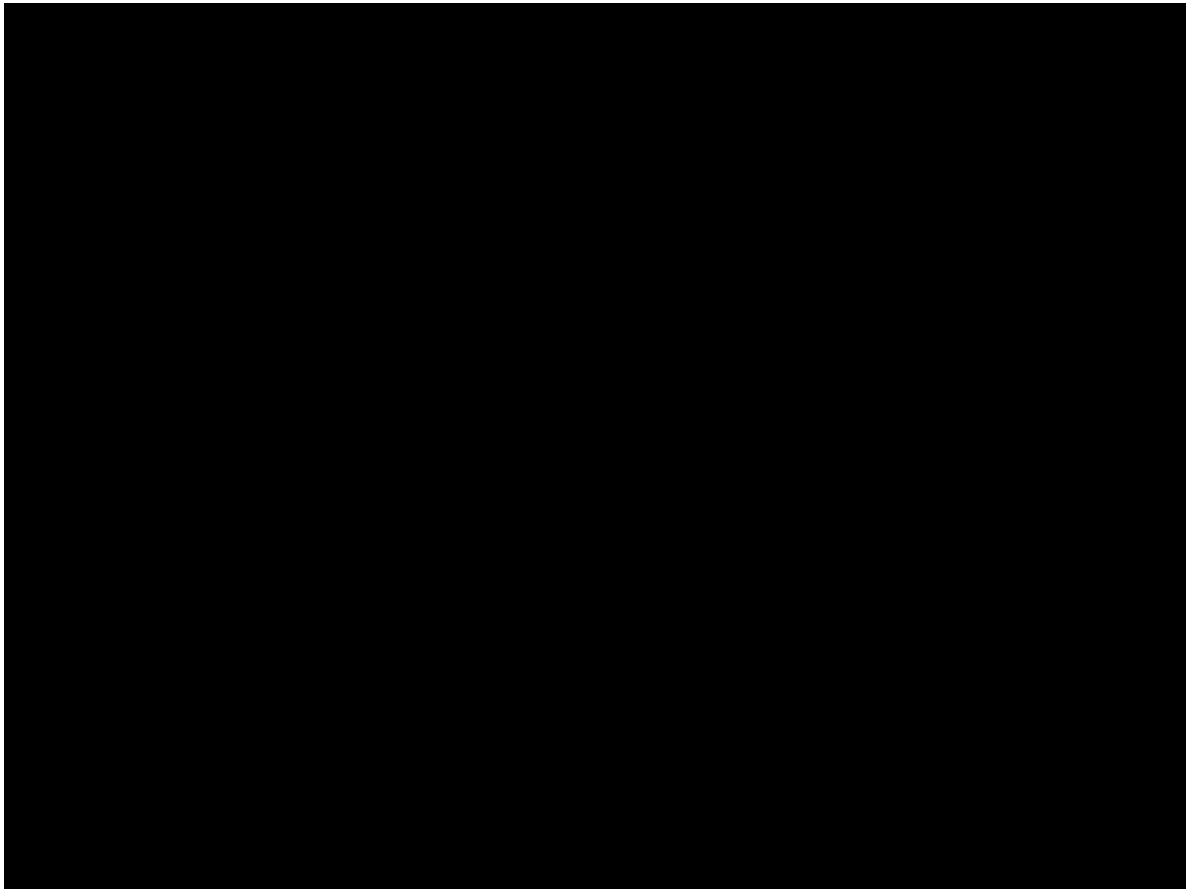
Function diagram

```
~/kars-adif-presentation  python3
Python 3.10.11 (main, Apr  4 2023, 22:10:32) [GCC 12.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from adif import save
>>> data = [{
... "call": "KI5WLJ",
... "freq": "14.324",
... "mode": "SSB",
... "qso_date": "20230624",
... "time_on": "1337",
... "station_callsign": "KT5TX"
... }]
>>>
>>> save(data, "test.adi")
>>> quit()

~/kars-adif-presentation  cat test.adi
```

```
File: test.adi

1  <adif_ver:5>3.1.4<programid:4>KARS<EOH>
2  <call:6>KI5WLJ<freq:6>14.324<mode:3>SSB<qso_date:8>20230624<time_on:4>1337<station_callsign:5>KT5TX<eor>
```

Testing our save function

Demonstration



More resources

The specification for ADIF is located at https://adif.org/adif

Python guides are available all over the internet, google “basic python guide” or “python help <something>”

You can run your own Python code at https://replit.com/l/python3

View the source code, presentation, or diagrams at https://ta.rdis.dev/ha/adif.html