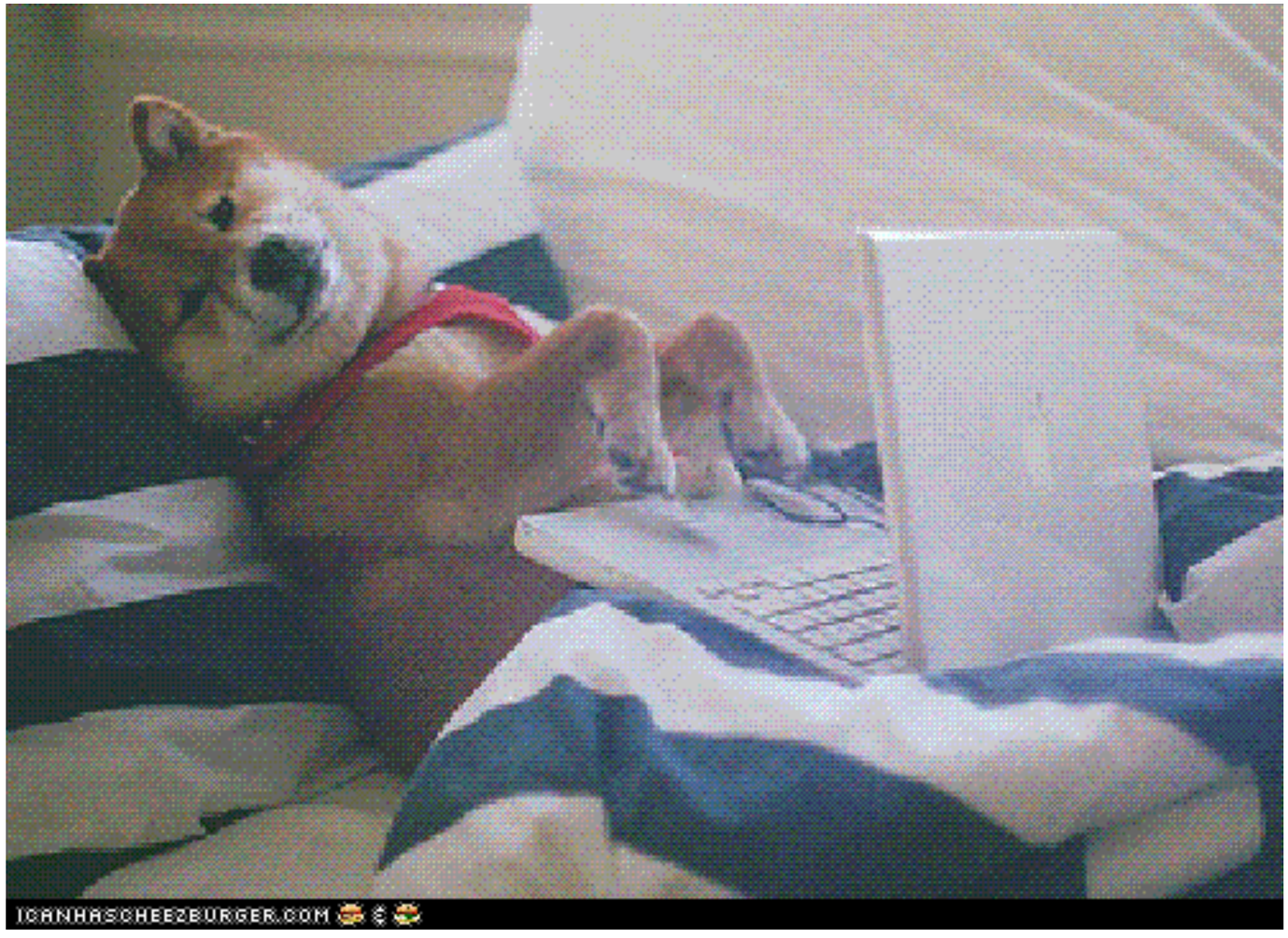
Write code for humans, not machines.

Narendran R
Backend Engineer @ MadStreetDen
Internet handle : @DudeWhoCode
website : dudewho.codes

Are you a "great" programmer?

Are you a "good" programmer?



- ***Variables***
- ***Functions***
- ***Loop/Indentation***
- ***Error handling***

Variables

```
temp = some_important_API.get('production_data')
```

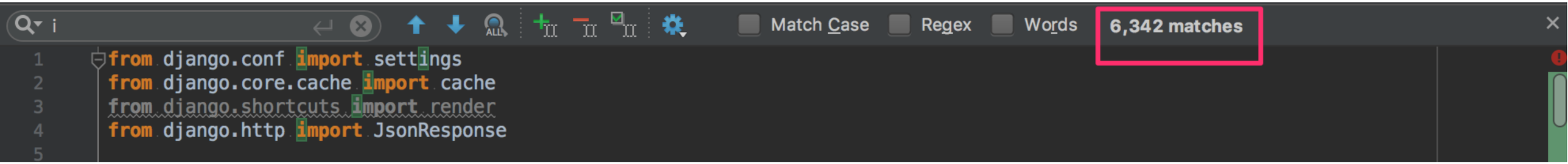
Our all time favourite **temp.**

Been there, done that. Hand up..

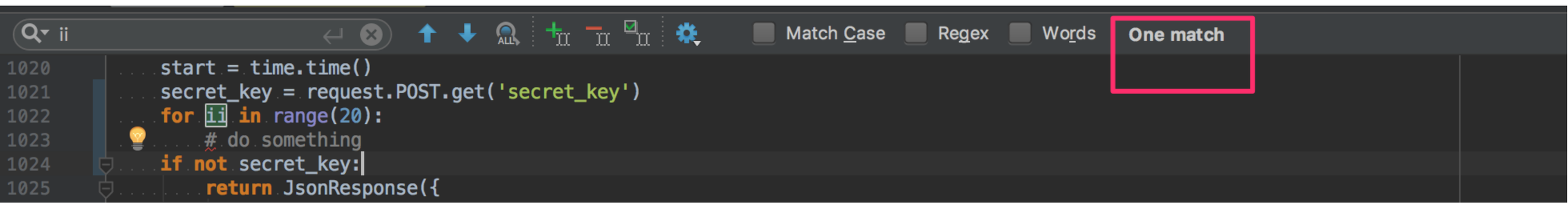
- Short and self explanatory
- For example, If you are selecting student names from DB

```
? = cursor.execute(query)
        res                    ❌
        tmp                    ❌
        data                   ❌
        students               ✅
        student_names          ✅
        names                  ✅
```

- Try not to use i, j, k in for loops and short scopes, you won't be able to find those variable names in large codebase.



- Use ii, jj, kk instead



```
temp = []
with open('product_catalog.txt', 'w') as file:
    for i in file:
        i = i.split(':')[-1].strip()
        temp.append(i)
```

(VS)

```
products = []
with open('product_catalog.txt', 'w') as f:
    for product in f:
        product = product.split(':')[-1].strip()
        products.append(product)
```

Functions/Methods

- Every function should have single responsibility

```
import json
import psycopg2
conn = psycopg2.connect("*your database credentials*")
cur = conn.cursor()

# Responsibilities : Read from a .txt file, check integrity of the
query, then insert to DB
def insert_db(queries_file):
    queries = open(queries_file)
    for query in queries:
        # check the query integrity
        if query.startswith('INSERT'):
            cur.execute(query)
            conn.commit()
        else:
            print('Invalid query')
            continue

if __name__ = '__main__':
    insert_db('queries.txt')
```

```
import json
import psycopg2
conn = psycopg2.connect(“*your database credentials*”)

# Responsibility : Return query one by one from a given .txt file
def load_queries(fname):
    queries = open(queries_file)
    for query in queries:
        yield query

# Responsibility : Check integrity of a given query
def check_query(query):
    if query.startswith(‘INSERT’):
        return True
    # All other query checks goes here
    return False

# Responsibility : Insert a given query into DB
def insert_db(query):
    cur = conn.execute(query)
    conn.commit()

# Responsibility : Put all functions from the script to a meaningful
use
def run(queryfile):
    for query in load_queries(queryfile):
        if check_query(query):
            insert_db(query)
        else:
            print(‘Invalid query’)
            continue


if __name__ = ‘__main__’:
    run(‘queries.txt')
```

Loops and Indentation

- Keep your nested loops to maximum 3 levels of indentation

```
def check_list_of_list_of_strings(data):
    if isinstance(data, list):
        for items in data:
            if isinstance(items, list):
                for item in items:
                    if isinstance(item, str):
                        if item.endswith('_name'):
                            return True
```

| 1 | 2| 3| 4| 5| 6| -> 6 levels of indentation 😱

```
def check_list_of_list_of_strings(data):
    if not isinstance(data, list):
        return False
    for items in data:
        if not isinstance(items, list):
            return False
        for item in items:
            if not (isinstance(item, str)):
                return False
            if not item.endswith('_name’):
                return False
```

| 1 | 2 | 3 | -> 3 levels of indentation

Error handling

The common anti-pattern

The silent killer

```
try:
    # do something
except:
    pass
```

- Allowing the error to silently pass through

The loud loser

```
try:
    # do something
except:
    raise
```

- The above try/except is meaningless as its going to raise an error with or without the try/except clause

Hey psssst.. !, Trust me you can't

```
bio = {‘first_name’: ‘Bruce’, ‘last_name’: 35, ‘age’: ‘Wayne',
‘super_power’: ‘being rich’}

try:
    fname = bio[‘first_name’]
    lname = bio[‘last_name’]
        full_name = fname + lname       # Throws TypeError
        super_power = bio[‘superPower’] # Throws KeyError
except Exception as e:
    log.Error(e)                        # You are logging only TypeError
        # How do you take an action without knowing the error?
```

```
bio = [‘first_name’: ‘Bruce’, ‘last_name’: 35, ‘age’: ‘Wayne',
‘superPower’: ‘being rich’]
try:
    fname = bio[‘first_name’]
    lname = bio[‘last_name’]
    full_name = fname + lname          # Throws TypeError
    super_power = bio[‘superPower’]    # Throws KeyError
    except KeyError:
    log.Error(‘key not found’)
        # Take appropriate actions as per the error
    except TypeError:
    log.Error(‘Unable to concatenate’)
        # Take appropriate actions as per the error
except Exception as e:
    log.Error(‘Unexpected Error’, e)
        # Expect the unexpected
```

Try not to swap :

```
except Exception as e:
    log.Error(‘Unexpected Error’, e)
    # Expect the unexpected
# Unreachable Code
except KeyError:
    log.Error(‘key not found’)
    # Take appropriate actions as per the error
except TypeError:
    log.Error(‘Unable to concatenate’)
    # Take appropriate actions as per the error
```

Summary

Variables :

- Short, meaningful, use underscores if necessary and no temp, temp1, data1 please.
- Easy readability

Functions :

- Give only one responsibility to a function
- Easy unit testing and readability

Summary

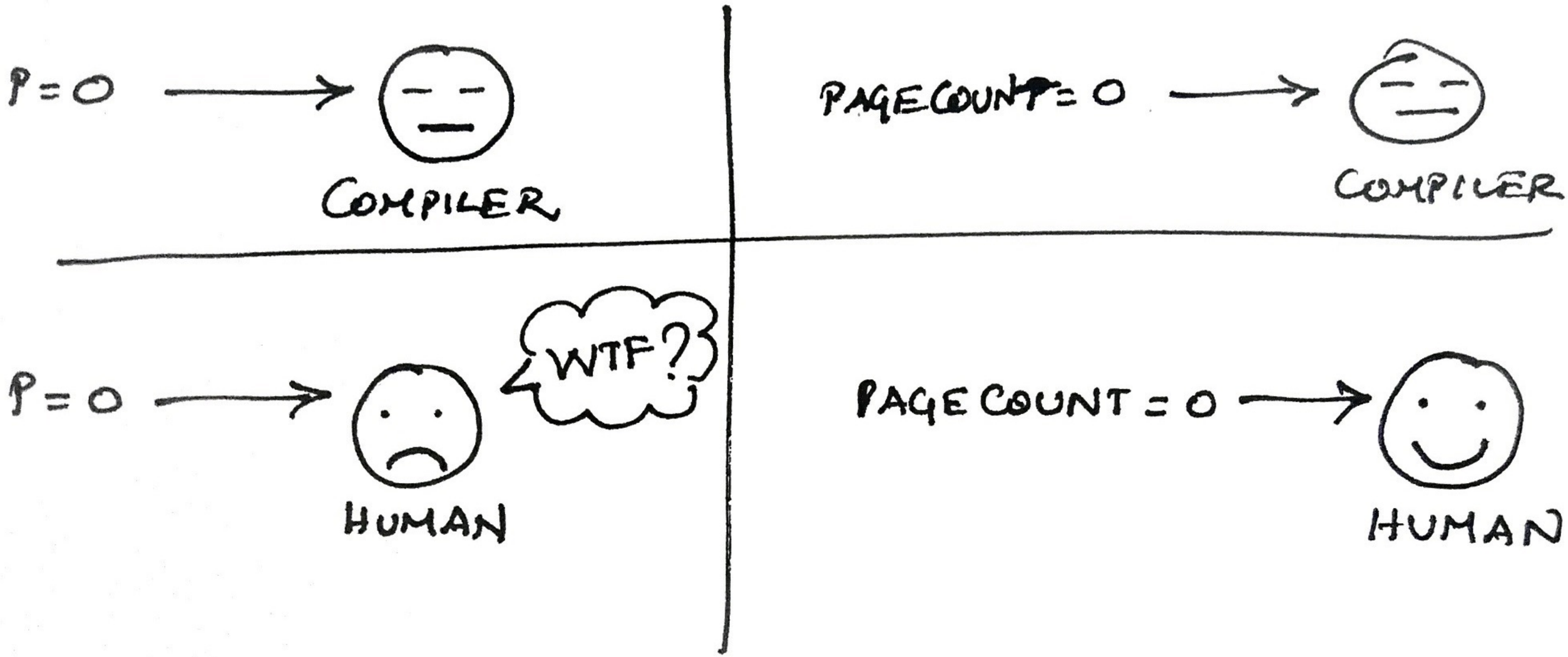
Loops :

- Don’t overdo the levels of nested loop

Error handling :

- Don’t try to catch them all
- No “raise” or “pass" inside except clause
- Know what error you are going to expect

Always remember ...



Whats next?

1. Go home, open the code that you wrote >6 months ago.
2. Read the code assuming that you are a new/junior programmer appointed to maintain above code.
3. Try not to punch yourself on the face.

- Just kidding

Whats really next?

1. Read PEP-8 and python anti-patterns.
2. Read open source code in your leisure time.
3. Develop good programming habits.
4. Make fellow humans happy.

I am not a great programmer, I am a good programmer with great habits.
- Kent beck

twitter, github :
@DudeWhoCode

slides :
dudewho.codes/talks