Google Python Course

Google Python ????

- [Course 1](#)
- [Dictionaries vs. Lists](#)
- [Classes and methods](#)
- [Examples](#)
- [Course 2](#)

Course 1

Naming rules and conventions

???????

When assigning names to objects, programmers adhere to a set of rules and conventions which help to standardize code and make it more accessible to everyone. Here are some naming rules and conventions that you should know:

- Names cannot contain spaces.
- Names may be a mixture of upper and lower case characters.
- Names can't start with a number but may contain numbers after the first character.
- Variable names and function names should be written in snake_case, which means that all letters are lowercase and words are separated using an underscore.
- Descriptive names are better than cryptic abbreviations because they help other programmers (and you) read and interpret your code. For example, student_name is better than sn. It may feel excessive when you write it, but when you return to your code you'll find it much easier to understand.

Common syntax errors

- Misspellings (????)
- Incorrect indentations (??????)
- Missing or incorrect key characters: (?????????)
  - Parenthetical types - (curved), [square], { curly } ???? - ??????????
  - Quote types - "straight-double" or 'straight-single', "curly-double" or 'curly-single' ????
  - Block introduction characters, like colons - : ??????
- Data type mismatches ????????
- Missing, incorrectly used, or misplaced Python reserved words ????????? Python ????
- Using the wrong case (uppercase/lowercase) - Python is a case-sensitive language ???????

Annotating variables by type

?????????

This has several benefits: It reduces the chance of common mistakes, helps in documenting your code for others to reuse, and allows integrated development software (IDEs) and other tools to give you better feedback.

How to annotate a variable:

```
a = 3                #a is an integer
captain = "Picard"    # type: str
captain: str = "Picard"

import typing
# Define a variable of type str
z: str = "Hello, world!"
# Define a variable of type int
x: int = 10
# Define a variable of type float
y: float = 1.23
# Define a variable of type list
list_of_numbers: typing.List[int] = [1, 2, 3]
# Define a variable of type tuple
tuple_of_numbers: typing.Tuple[int, int, int] = (1, 2, 3)
# Define a variable of type dict
dictionary: typing.Dict[str, int] = {"key1": 1, "key2": 2}
# Define a variable of type set
set_of_numbers: typing.Set[int] = {1, 2, 3}
```

Data type conversions

Implicit vs explicit conversion ?? vs ????

Implicit conversion is where the interpreter helps us out and **automatically converts one data type into another**, without having to explicitly tell it to do so.

Example:

```
# Converting integer into a float
print(7+8.5)
```

Explicit conversion is where we **manually convert from one data type to another** by calling the **relevant function** for the data type we want to convert to.

We used this in our video example when we wanted to print a number alongside some text. Before we could do that, we needed to call the *str()* function to convert the number into a string.

- **str()** - converts a value (often numeric) to a string data type

- **int()** - converts a value (usually a float) to an integer data type
- **float()** - converts a value (usually an integer) to a float data type

Example:

```
# Convert a number into a string
base = 6
height = 3
area = (base*height)/2
print("The area of the triangle is: " + str(area))
```

Operators

Arithmetic operators

- `//` ???? (Floor division operator)
- `%` ???? (Modulo operator)
- `**` ??

Example for // & %

```
# even: 󠀀󠀀
def is_even(number):
    if number % 2 == 0:
        return True
    return False
#This code has no ouput
```

```
def calculate_storage(filesize):
    block_size = 4096
    # Use floor division to calculate how many blocks are fully occupied
    full_blocks = filesize // block_size
    # Use the modulo operator to check whether there's any remainder
    partial_block_remainder = filesize % block_size
    # Depending on whether there's a remainder or not, return
    # the total number of bytes required to allocate enough blocks
    # to store your data.
    if partial_block_remainder > 0:
        return (full_blocks + 1) * block_size
    return full_blocks * block_size
```

```
print(calculate_storage(1))    # Should be 4096
print(calculate_storage(4096)) # Should be 4096
print(calculate_storage(4097)) # Should be 8192
print(calculate_storage(6000)) # Should be 8192
```

Comparison operators

Symbol	Name	Expression	Description
==	Equality operator	a == b	a is equal to b
!=	Not equal to operator	a != b	a is **not** equal to b
>	Greater than operator	a > b	a is larger than b
>=	Greater than or equal to operator	a >= b	a is larger than or equal to b
<	Less than operator	a < b	a is smaller than b
<=	Less than or equal to operator	a <= b	a is smaller than or equal to b

Good coding style

- **Create a reusable function** - Replace duplicate code with one reusable function to make the code easier to read and repurpose.
- **Refactor code** - Update code so that it is self-documenting and the intent of the code is clear.
- **Add comments** - Adding comments is part of creating self-documenting code. Using comments allows you to leave notes to yourself and/or other programmers to make the purpose of the code clear.
  ????????????????????????????????/???????????????????????????

Loops

While Loops

```
multiplier = 1
result = multiplier * 5
while result <= 50:
    print(result)
    multiplier += 1
    result = multiplier * 5
```

```
print("Done")
```

Common errors in Loops

- **Failure to initialize variables**. Make sure all the variables used in the loop's condition are initialized before the loop.
- **Unintended infinite loops**. Make sure that the body of the loop modifies the variables used in the condition, so that the loop will eventually end for all possible values of the variables. You can often prevent an infinite loop by using the break keyword or by adding end criteria to the condition part of the *while* loop.

For Loops

```
friends = ['Taylor', 'Alex', 'Pat', 'Eli']
for friend in friends:
    print("Hi " + friend)
```

```
# °F to °C
def to_celsius(x):
  return (x-32)*5/9


for x in range(0,101,10):
  print(x, to_celsius(x))
```

```
for number in range(1, 6+1, 2):
    print(number * 3)


# The loop should print 3, 9, 15
```

Nested for Loops

??? for ??

```
# home_team 主場, away_team 客場
teams = [ 'Dragons', 'Wolves', 'Pandas', 'Unicorns']
for home_team in teams:
  for away_team in teams:
    if home_team != away_team:
      print(home_team + " vs " + away_team)
```

List comprehensions

?????: `[x for x in sequence if condition]`

```python
# with for loop
numbers = [1, 2, 3, 4, 5]
squared_numbers = [x ** 2 for x in numbers]
print(squared_numbers)
```

```python
# with for loop and if
sequence = range(10)
new_list = [x for x in sequence if x % 2 == 0]
```

Recursive function

???? Use cases

1. Goes through a bunch of directories in your computer and calculates how many files are contained in each.
2. Review groups in Active Directory.

```python
'''
def recursive_function(parameters):
    if base_case_condition(parameters):
        return base_case_value
    recursive_function(modified_parameters)
'''
def factorial(n):
  if n < 2:
    return 1
  return n * factorial(n-1)
```

```python
def factorial(n):
  print("Factorial called with " + str(n))
  if n < 2:
    print("Returning 1")
    return 1
  result = n * factorial(n-1)
  print("Returning " + str(result) + " for factorial of " + str(n))
  return result

factorial(4)
```

Types of iterables

- **String**: ??? (sequential)???? (immutable) ????????
- **List**: ??? (sequential)??? (mutable) ??????????
- **Dictionary**: ??????? key:value ??????
- **Tuple**: ??? (sequential)???? (immutable) ??????????
- **Set**: ???? (unordered)???? (unique) ???????

Resources

Naming rules and conventions

- [PEP 8 – Style Guide for Python Code](#)

Annotating variables by type

- [Built-in Types — Python 3.13.0 documentation](#)

Dictionaries vs. Lists

Dictionaries are similar to lists, but there are a few differences:

Both dictionaries and lists:

- are used to organize elements into collections;
- are used to initialize a new dictionary or list, use empty brackets;
- can iterate through the items or elements in the collection; and
- can use a variety of methods and operations to create and change the collections, like removing and inserting items or elements.

Dictionaries only:

- are unordered sets;
- have keys that can be a variety of data types, including strings, integers, floats, tuples;.
- can access dictionary values by keys;
- use square brackets inside curly brackets { [ ] };
- use colons between the key and the value(s);
- use commas to separate each key group and each value within a key group;
- make it quicker and easier for a Python interpreter to find specific elements, as compared to a list.

```
pet_dictionary = {"dogs": ["Yorkie", "Collie", "Bulldog"], "cats": ["Persian", "Scottish Fold", "Siberian"], "rabbits": ["Angora", "Holland Lop", "Harlequin"]}


print(pet_dictionary.get("dogs", 0))
# Should print ['Yorkie', 'Collie', 'Bulldog']
```

Lists only:

- are ordered sets;
- access list elements by index positions;
- require that these indices be integers;
- use square brackets [ ];
- use commas to separate each list element.

```python
pet_list  = ["Yorkie", "Collie", "Bulldog", "Persian", "Scottish Fold", "Siberian", "Angora", "Holland Lop",
"Harlequin"]


print(pet_list[0:3])
# Should print ['Yorkie', 'Collie', 'Bulldog']
```

Classes and methods

Defining classes and methods

```
class ClassName:
    def method_name(self, other_parameters):
        body_of_method
```

Special methods

- Special methods start and end with __.
- Special methods have specific names, like __init__ for the constructor or __str__ for the conversion to string.
- The methods __str__ and __repr__ allow you to define human-readable and unambiguous string representations of your objects, respectively.
- By defining methods like __eq__, __ne__, __lt__, __gt__, __le__, and __ge__, you can control how *objects* of your *class* are compared.

With the __init__ method:

???????????????? self.XXX

```
class Apple:
    def __init__(self, color, flavor):
        self.color = color
        self.flavor = flavor


honeycrisp = Apple("red", "sweet")
fuji = Apple("red", "tart")
print(honeycrisp.flavor)
print(fuji.flavor)
```

With the __str__ method:

When you print() something, Python calls the object's __str__() method and outputs whatever that method returns

```
class Apple:
    def __init__(self, color, flavor):
        self.color = color
        self.flavor = flavor


    def __str__(self):
        return "an apple which is {} and {}".format(self.color, self.flavor)


honeycrisp = Apple("red", "sweet")
print(honeycrisp)


# prints "an apple which is red and sweet"
```

With the custom method

```
class Triangle:
    def __init__(self, base, height):
        self.base = base
        self.height = height
    def area(self):
        return 0.5 * self.base * self.height
    def __add__(self, other):
        return self.area() + other.area()


triangle1 = Triangle(10, 5)
triangle2 = Triangle(6, 8)
print("The area of triangle 1 is", triangle1.area())
print("The area of triangle 2 is", triangle2.area())
print("The area of both triangles is", triangle1 + triangle2)
```

Examples

??????

- Custom Class
- Dictionary/Set/List Data
- Set Methods

```python
def get_event_date(event):
  return event.date


def current_users(events):
  events.sort(key=get_event_date)
  machines = {}
  for event in events:
    if event.machine not in machines:
      machines[event.machine] = set()
    if event.type == "login":
      machines[event.machine].add(event.user)
    elif event.type == "logout":
      machines[event.machine].remove(event.user)
  return machines


def generate_report(machines):
  for machine, users in machines.items():
    if len(users) > 0:
      user_list = ", ".join(users)
      print("{}: {}".format(machine, user_list))

class Event:
  def __init__(self, event_date, event_type, machine_name, user):
    self.date = event_date
    self.type = event_type
    self.machine = machine_name
    self.user = user


events = [
```

```
  Event('2020-01-21 12:45:46', 'login', 'myworkstation.local', 'jordan'),
  Event('2020-01-22 15:53:42', 'logout', 'webserver.local', 'jordan'),
  Event('2020-01-21 18:53:21', 'login', 'webserver.local', 'lane'),
  Event('2020-01-22 10:25:34', 'logout', 'myworkstation.local', 'jordan'),
  Event('2020-01-21 08:20:01', 'login', 'webserver.local', 'jordan'),
  Event('2020-01-23 11:24:35', 'login', 'mailserver.local', 'chris'),
]


users = current_users(events)
print(users)
# Output: {'webserver.local': {'lane'}, 'myworkstation.local': set(), 'mailserver.local': {'chris'}}


generate_report(users)
# Output:
# webserver.local: lane
# mailserver.local: chris
```

?? Syslog

- dictionary.get()
- re.search()
- with open() as f

```
import re
import sys


logfile = sys.argv[1]
usernames = {}
with open(logfile) as f:
  for line in f:
    if "CRON" not in line:
      continue
    pattern = r"USER \((\w+)\)$"
    result = re.search(pattern, line)

    if result is None:
      continue
    name = result[1]
    usernames[name] = usernames.get(name, 0) + 1
```

```
    print(usernames)
```

???

fishy.log:

```
July 31 02:25:52 mycomputername system[41921]: WARN Failed to start CPU thread[39016]

July 31 02:34:37 mycomputername kernel[32280]: INFO Loading...

July 31 02:36:44 mycomputername NetworkManager[90289]: WARN Failed to start CPU thread[39016]

July 31 02:39:01 mycomputername CRON[89330]: ERROR Unable to perform package upgrade

July 31 02:45:39 mycomputername utility[57387]: INFO Access permitted

July 31 02:58:44 mycomputername process[44707]: WARN Computer needs to be turned off and on again

July 31 02:59:35 mycomputername system[55024]: WARN Packet loss

July 31 03:09:30 mycomputername kernel[40705]: ERROR The cake is a lie!

July 31 03:23:16 mycomputername cacheclient[57185]: INFO Checking process [16121]

July 31 03:26:56 mycomputername cacheclient[90154]: INFO Healthy resource usage

July 31 03:28:52 mycomputername CRON[55441]: INFO Loading...

July 31 03:29:34 mycomputername dhcpclient[69232]: ERROR Unable to download more RAM

July 31 03:34:41 mycomputername NetworkManager[14120]: ERROR 404 error not found

July 31 03:36:26 mycomputername dhcpclient[79731]: ERROR The cake is a lie!

July 31 03:38:24 mycomputername CRON[92141]: INFO Access permitted

July 31 03:40:00 mycomputername dhcpclient[40114]: INFO Starting sync

July 31 03:42:45 mycomputername utility[53726]: INFO I'm sorry Dave. I'm afraid I can't do that

July 31 03:47:07 mycomputername NetworkManager[63805]: WARN Please reboot user

July 31 04:09:16 mycomputername CRON[52593]: WARN PC Load Letter

July 31 04:11:32 mycomputername CRON[51253]: ERROR: Failed to start CRON job due to script syntax error.
Inform the CRON job owner!

July 31 04:11:32 mycomputername jam_tag=psim[84082]: ERROR ID: 10t

July 31 04:12:05 mycomputername utility[63418]: INFO Successfully connected

July 31 04:14:22 mycomputername utility[53225]: ERROR I am error

July 31 04:31:00 mycomputername NetworkManager[23060]: ERROR Out of yellow ink, specifically, even though
you want grayscale
```

find_error.py

Usage: `./find_error.py fishy.log`

```
import sys

import os

import re
```

```python
def error_search(log_file):
    error = input("What is the error? ")
    returned_errors = []

    with open(log_file, mode='r', encoding='UTF-8') as file:
        for log in file.readlines():
            error_patterns = ["error"]
            for i in range(len(error.split(' '))):
                error_patterns.append(r"{}".format(error.split(' ')[i].lower()))

                if all(re.search(error_pattern, log.lower()) for error_pattern in error_patterns):
                    returned_errors.append(log)

        file.close()
    return returned_errors

def file_output(returned_errors):
    with open(os.path.expanduser('~') + '/data/errors_found.log', 'w') as file:
        for error in returned_errors:
            file.write(error)

    file.close()

if __name__ == "__main__":
    log_file = sys.argv[1]
    returned_errors = error_search(log_file)
    file_output(returned_errors)
    sys.exit(0)
```

?? Syslog 2

syslog.log :

```
Jan 31 00:09:39 ubuntu.local ticky: INFO Created ticket [#4217] (mdouglas)
Jan 31 00:16:25 ubuntu.local ticky: INFO Closed ticket [#1754] (noel)
Jan 31 00:21:30 ubuntu.local ticky: ERROR The ticket was modified while updating (breee)
Jan 31 00:44:34 ubuntu.local ticky: ERROR Permission denied while closing ticket (ac)
Jan 31 01:00:50 ubuntu.local ticky: INFO Commented on ticket [#4709] (blossom)
Jan 31 01:29:16 ubuntu.local ticky: INFO Commented on ticket [#6518] (rr.robinson)
```

Jan 31 01:33:12 ubuntu.local ticky: ERROR Tried to add information to closed ticket (mcintosh)

Jan 31 01:43:10 ubuntu.local ticky: ERROR Tried to add information to closed ticket (jackowens)

Jan 31 01:49:29 ubuntu.local ticky: ERROR Tried to add information to closed ticket (mdouglas)

Jan 31 02:30:04 ubuntu.local ticky: ERROR Timeout while retrieving information (oren)

Jan 31 02:55:31 ubuntu.local ticky: ERROR Ticket doesn't exist (xlg)

Jan 31 03:05:35 ubuntu.local ticky: ERROR Timeout while retrieving information (ahmed.miller)

Jan 31 03:08:55 ubuntu.local ticky: ERROR Ticket doesn't exist (blossom)

Jan 31 03:39:27 ubuntu.local ticky: ERROR The ticket was modified while updating (bpacheco)

Jan 31 03:47:24 ubuntu.local ticky: ERROR Ticket doesn't exist (enim.non)

Jan 31 04:30:04 ubuntu.local ticky: ERROR Permission denied while closing ticket (rr.robinson)

Jan 31 04:31:49 ubuntu.local ticky: ERROR Tried to add information to closed ticket (oren)

Jan 31 04:32:49 ubuntu.local ticky: ERROR Timeout while retrieving information (mcintosh)

Jan 31 04:44:23 ubuntu.local ticky: ERROR Timeout while retrieving information (ahmed.miller)

Jan 31 04:44:46 ubuntu.local ticky: ERROR Connection to DB failed (jackowens)

Jan 31 04:49:28 ubuntu.local ticky: ERROR Permission denied while closing ticket (flavia)

Jan 31 05:12:39 ubuntu.local ticky: ERROR Tried to add information to closed ticket (oren)

Jan 31 05:18:45 ubuntu.local ticky: ERROR Tried to add information to closed ticket (sri)

Jan 31 05:23:14 ubuntu.local ticky: INFO Commented on ticket [#1097] (breee)

Jan 31 05:35:00 ubuntu.local ticky: ERROR Connection to DB failed (nonummy)

Jan 31 05:45:30 ubuntu.local ticky: INFO Created ticket [#7115] (noel)

Jan 31 05:51:30 ubuntu.local ticky: ERROR The ticket was modified while updating (flavia)

Jan 31 05:57:46 ubuntu.local ticky: INFO Commented on ticket [#2253] (nonummy)

Jan 31 06:12:02 ubuntu.local ticky: ERROR Connection to DB failed (oren)

Jan 31 06:26:38 ubuntu.local ticky: ERROR Timeout while retrieving information (xlg)

Jan 31 06:32:26 ubuntu.local ticky: INFO Created ticket [#7298] (ahmed.miller)

Jan 31 06:36:25 ubuntu.local ticky: ERROR Timeout while retrieving information (flavia)

Jan 31 06:57:00 ubuntu.local ticky: ERROR Connection to DB failed (jackowens)

Jan 31 06:59:57 ubuntu.local ticky: INFO Commented on ticket [#7255] (oren)

Jan 31 07:59:56 ubuntu.local ticky: ERROR Ticket doesn't exist (flavia)

Jan 31 08:01:40 ubuntu.local ticky: ERROR Tried to add information to closed ticket (jackowens)

Jan 31 08:03:19 ubuntu.local ticky: INFO Closed ticket [#1712] (britanni)

Jan 31 08:22:37 ubuntu.local ticky: INFO Created ticket [#2860] (mcintosh)

Jan 31 08:28:07 ubuntu.local ticky: ERROR Timeout while retrieving information (montanap)

Jan 31 08:49:15 ubuntu.local ticky: ERROR Permission denied while closing ticket (britanni)

Jan 31 08:50:50 ubuntu.local ticky: ERROR Permission denied while closing ticket (montanap)

Jan 31 09:04:27 ubuntu.local ticky: ERROR Tried to add information to closed ticket (noel)

Jan 31 09:15:41 ubuntu.local ticky: ERROR Timeout while retrieving information (oren)

Jan 31 09:18:47 ubuntu.local ticky: INFO Commented on ticket [#8385] (mdouglas)

Jan 31 09:28:18 ubuntu.local ticky: INFO Closed ticket [#2452] (jackowens)

Jan 31 09:41:16 ubuntu.local ticky: ERROR Connection to DB failed (ac)

Jan 31 10:11:35 ubuntu.local ticky: ERROR Timeout while retrieving information (blossom)

Jan 31 10:21:36 ubuntu.local ticky: ERROR Permission denied while closing ticket (montanap)

Jan 31 11:04:02 ubuntu.local ticky: ERROR Tried to add information to closed ticket (breee)

Jan 31 11:19:37 ubuntu.local ticky: ERROR Connection to DB failed (sri)

Jan 31 11:22:06 ubuntu.local ticky: ERROR Timeout while retrieving information (montanap)

Jan 31 11:31:34 ubuntu.local ticky: ERROR Permission denied while closing ticket (ahmed.miller)

Jan 31 11:40:25 ubuntu.local ticky: ERROR Connection to DB failed (mai.hendrix)

Jan 31 11:47:07 ubuntu.local ticky: INFO Commented on ticket [#4562] (ac)

Jan 31 11:58:33 ubuntu.local ticky: ERROR Tried to add information to closed ticket (ahmed.miller)

Jan 31 12:00:17 ubuntu.local ticky: INFO Created ticket [#7897] (kirknixon)

Jan 31 12:02:49 ubuntu.local ticky: ERROR Permission denied while closing ticket (mai.hendrix)

Jan 31 12:20:23 ubuntu.local ticky: ERROR Connection to DB failed (kirknixon)

Jan 31 12:20:40 ubuntu.local ticky: ERROR Ticket doesn't exist (flavia)

Jan 31 12:24:32 ubuntu.local ticky: INFO Created ticket [#5784] (sri)

Jan 31 12:50:10 ubuntu.local ticky: ERROR Permission denied while closing ticket (blossom)

Jan 31 12:58:16 ubuntu.local ticky: ERROR Tried to add information to closed ticket (nonummy)

Jan 31 13:08:10 ubuntu.local ticky: INFO Closed ticket [#8685] (rr.robinson)

Jan 31 13:48:45 ubuntu.local ticky: ERROR The ticket was modified while updating (breee)

Jan 31 14:13:00 ubuntu.local ticky: INFO Commented on ticket [#4225] (noel)

Jan 31 14:38:50 ubuntu.local ticky: ERROR The ticket was modified while updating (enim.non)

Jan 31 14:41:18 ubuntu.local ticky: ERROR Timeout while retrieving information (xlg)

Jan 31 14:45:55 ubuntu.local ticky: INFO Closed ticket [#7948] (noel)

Jan 31 14:50:41 ubuntu.local ticky: INFO Commented on ticket [#8628] (noel)

Jan 31 14:56:35 ubuntu.local ticky: ERROR Tried to add information to closed ticket (noel)

Jan 31 15:27:53 ubuntu.local ticky: ERROR Ticket doesn't exist (blossom)

Jan 31 15:28:15 ubuntu.local ticky: ERROR Permission denied while closing ticket (enim.non)

Jan 31 15:44:25 ubuntu.local ticky: INFO Closed ticket [#7333] (enim.non)

Jan 31 16:17:20 ubuntu.local ticky: INFO Commented on ticket [#1653] (noel)

Jan 31 16:19:40 ubuntu.local ticky: ERROR The ticket was modified while updating (mdouglas)

Jan 31 16:24:31 ubuntu.local ticky: INFO Created ticket [#5455] (ac)

Jan 31 16:35:46 ubuntu.local ticky: ERROR Timeout while retrieving information (oren)

Jan 31 16:53:54 ubuntu.local ticky: INFO Commented on ticket [#3813] (mcintosh)

Jan 31 16:54:18 ubuntu.local ticky: ERROR Connection to DB failed (bpacheco)

Jan 31 17:15:47 ubuntu.local ticky: ERROR The ticket was modified while updating (mcintosh)

Jan 31 17:29:11 ubuntu.local ticky: ERROR Connection to DB failed (oren)

Jan 31 17:51:52 ubuntu.local ticky: INFO Closed ticket [#8604] (mcintosh)

Jan 31 18:09:17 ubuntu.local ticky: ERROR The ticket was modified while updating (noel)

Jan 31 18:43:01 ubuntu.local ticky: ERROR Ticket doesn't exist (nonummy)

Jan 31 19:00:23 ubuntu.local ticky: ERROR Timeout while retrieving information (blossom)

Jan 31 19:20:22 ubuntu.local ticky: ERROR Timeout while retrieving information (mai.hendrix)

Jan 31 19:59:06 ubuntu.local ticky: INFO Created ticket [#6361] (enim.non)

Jan 31 20:02:41 ubuntu.local ticky: ERROR Timeout while retrieving information (xlg)

Jan 31 20:21:55 ubuntu.local ticky: INFO Commented on ticket [#7159] (ahmed.miller)

Jan 31 20:28:26 ubuntu.local ticky: ERROR Connection to DB failed (breee)

Jan 31 20:35:17 ubuntu.local ticky: INFO Created ticket [#7737] (nonummy)

Jan 31 20:48:02 ubuntu.local ticky: ERROR Connection to DB failed (mdouglas)

Jan 31 20:56:58 ubuntu.local ticky: INFO Closed ticket [#4372] (oren)

Jan 31 21:00:23 ubuntu.local ticky: INFO Commented on ticket [#2389] (sri)

Jan 31 21:02:06 ubuntu.local ticky: ERROR Connection to DB failed (breee)

Jan 31 21:20:33 ubuntu.local ticky: INFO Closed ticket [#3297] (kirknixon)

Jan 31 21:29:24 ubuntu.local ticky: ERROR The ticket was modified while updating (blossom)

Jan 31 22:58:55 ubuntu.local ticky: INFO Created ticket [#2461] (jackowens)

Jan 31 23:25:18 ubuntu.local ticky: INFO Closed ticket [#9876] (blossom)

Jan 31 23:35:40 ubuntu.local ticky: INFO Created ticket [#5896] (mcintosh)

ticky_check.py

Usage: `./ticky_check.py`

```python
#!/usr/bin/env python3
import sys
import re
import operator
import csv

# Dict: Count number of entries for each user
per_user = {}  # Splitting between INFO and ERROR
# Dict: Number of different error messages
errors = {}

# * Read file and create dictionaries
with open('syslog.log') as file:
    # read each line
    for line in file.readlines():
        # regex search
        # * Sample Line of log file
        # "May 27 11:45:40 ubuntu.local ticky: INFO: Created ticket [#1234] (username)"
        match = re.search(
            r"ticky: ([\w+]*):? ([\w' ]*)[\[[#0-9]*\]?]? ?\((.*)\)$", line)
        code, error_msg, user = match.group(1), match.group(2), match.group(3)
```

```python
        # Populates error dict with ERROR messages from log file
        if error_msg not in errors.keys():
            errors[error_msg] = 1
        else:
            errors[error_msg] += 1
        # Populates per_user dict with users and default values
        if user not in per_user.keys():
            per_user[user] = {}
            per_user[user]['INFO'] = 0
            per_user[user]['ERROR'] = 0
        # Populates per_user dict with users logs entry
        if code == 'INFO':
            if user not in per_user.keys():
                per_user[user] = {}
                per_user[user]['INFO'] = 0
            else:
                per_user[user]["INFO"] += 1
        elif code == 'ERROR':
            if user not in per_user.keys():
                per_user[user] = {}
                per_user[user]['INFO'] = 0
            else:
                per_user[user]['ERROR'] += 1


# Sorted by VALUE (Most common to least common)
errors_list = sorted(errors.items(), key=operator.itemgetter(1), reverse=True)

# Sorted by USERNAME
per_user_list = sorted(per_user.items(), key=operator.itemgetter(0))

file.close()
# Insert at the beginning of the list
errors_list.insert(0, ('Error', 'Count'))
per_user_list.insert(0, ('Username', {'INFO': 'INFO', 'ERROR': 'ERROR'}))

# * Create CSV file user_statistics
with open('user_statistics.csv', 'w', newline='') as user_csv:
    for key, value in per_user_list:
```

```
        user_csv.write(str(key) + ',' +
                str(value['INFO']) + ',' + str(value['ERROR'])+'\n')


# * Create CSV error_message
with open('error_message.csv', 'w', newline='') as error_csv:
    for key, value in errors_list:
        error_csv.write(str(key) + ',' + str(value) + '\n')
```

csv_to_html.py

Usage: `./csv_to_html.py user_statistics.csv /var/www/html/<html-filename>.html`

```
#!/usr/bin/env python3


import sys
import csv
import os

def process_csv(csv_file):
    """Turn the contents of the CSV file into a list of lists"""
    print("Processing {}".format(csv_file))
    with open(csv_file,"r") as datafile:
        data = list(csv.reader(datafile))
    return data

def data_to_html(title, data):
    """Turns a list of lists into an HTML table"""

    # HTML Headers
    html_content = """
<html>
<head>
<style>
table {
    width: 25%;
    font-family: arial, sans-serif;
    border-collapse: collapse;
}
```

```
tr:nth-child(odd) {
    background-color: #dddddd;
}

td, th {
    border: 1px solid #dddddd;
    text-align: left;
    padding: 8px;
}
</style>
</head>
<body>
"""


    # Add the header part with the given title
    html_content += "<h2>{}</h2><table>".format(title)


    # Add each row in data as a row in the table
    # The first line is special and gets treated separately
    for i, row in enumerate(data):
        html_content += "<tr>"
        for column in row:
            if i == 0:
                html_content += "<th>{}</th>".format(column)
            else:
                html_content += "<td>{}</td>".format(column)
        html_content += "</tr>"


    html_content += """</tr></table></body></html>"""
    return html_content



def write_html_file(html_string, html_file):


    # Making a note of whether the html file we're writing exists or not
    if os.path.exists(html_file):
        print("{} already exists. Overwriting...".format(html_file))


    with open(html_file,'w') as htmlfile:
```

```python
        htmlfile.write(html_string)
    print("Table succesfully written to {}".format(html_file))


def main():
    """Verifies the arguments and then calls the processing function"""
    # Check that command-line arguments are included
    if len(sys.argv) < 3:
        print("ERROR: Missing command-line argument!")
        print("Exiting program...")
        sys.exit(1)


    # Open the files
    csv_file = sys.argv[1]
    html_file = sys.argv[2]


    # Check that file extensions are included
    if ".csv" not in csv_file:
        print('Missing ".csv" file extension from first command-line argument!')
        print("Exiting program...")
        sys.exit(1)


    if ".html" not in html_file:
        print('Missing ".html" file extension from second command-line argument!')
        print("Exiting program...")
        sys.exit(1)


    # Check that the csv file exists
    if not os.path.exists(csv_file):
        print("{} does not exist".format(csv_file))
        print("Exiting program...")
        sys.exit(1)


    # Process the data and turn it into an HTML
    data = process_csv(csv_file)
    title = os.path.splitext(os.path.basename(csv_file))[0].replace("_", " ").title()
    html_string = data_to_html(title, data)
    write_html_file(html_string, html_file)


if __name__ == "__main__":
    main()
```

Course 2

Understanding Slowness

Slow Web Server

ab - Apache benchmark tool

    ab -n 500 site.example.com

Profiling - Improving the code

Profiling ?????????????????????????????????????? IT ???????Profile ??????????????? Profiling ??????????????????????? Profiling ???????????????????????????????

A profiler is a tool that measures the resources that our code is using, giving us a better understanding of what's going on.

- gprof : For C program
- cProfile : For Python program
- pprofile3 + kcachegrind(GUI) : For Python program
- Flat, Call-graph, and Input-sensitive are integral to debugging
- timeit (python module) : Measure execution time of small code snippets

Parallelizing operations

- [Speed Up Your Python Program With Concurrency – Real Python](#)

Python modules

- threading
- asyncio
- future

Concurrency for I/O-bound tasks

Python has two main approaches to implementing concurrency: threading and asyncio.

1. Threading is an efficient method for overlapping waiting times. This makes it well-suited for tasks involving many I/O operations, such as file I/O or network operations that spend significant time waiting. There are however some limitations with threading in Python due to the Global Interpreter Lock (GIL), which can limit the utilization of multiple cores.
2. Alternatively, asyncio is another powerful Python approach for concurrency that uses the event loop to manage task switching. Asyncio provides a higher degree of control, scalability, and power than threading for I/O-bound tasks. Any application that involves reading and writing data can benefit from it, since it speeds up I/O-based programs. Additionally, asyncio operates cooperatively and bypasses GIL limitations, enabling better performance for I/O-bound tasks.

Python supports concurrent execution through both threading and asyncio; however, asyncio is particularly beneficial for I/O-bound tasks, making it significantly faster for applications that read and write a lot of data.

Parallelism for CPU-bound tasks

Parallelism is a powerful technique for programs that heavily rely on the CPU to process large volumes of data constantly. It's especially useful for CPU-bound tasks like calculations, simulations, and data processing.

Instead of interleaving and executing tasks concurrently, parallelism enables multiple tasks to run simultaneously on multiple CPU cores. This is crucial for applications that require significant CPU resources to handle intense computations in real-time.

Multiprocessing libraries in Python facilitate parallel execution by distributing tasks across multiple CPU cores. It ensures performance by giving each process its own Python interpreter and memory space. It allows CPU-bound Python programs to process data more efficiently by giving each process its own Python interpreter and memory space; this eliminates conflicts and slowdowns caused by sharing resources. Having said that, you should also remember that when running multiple tasks simultaneously, you need to manage resources carefully.

Combining concurrency and parallelism

Combining concurrency and parallelism can improve performance. In certain complex applications with both I/O-bound and CPU-bound tasks, you can use asyncio for concurrency and multiprocessing for parallelism.

With asyncio, you make I/O-bound tasks more efficient as the program can do other things while waiting for file operations.

On the other hand, multiprocessing allows you to distribute CPU-bound computations, like heavy calculations, across multiple processors for faster execution.

By combining these techniques, you can create a well-optimized and responsive program. Your I/O-bound tasks benefit from concurrency, while CPU-bound tasks leverage parallelism.

psutil

```
# Installation
pip3 install psutil
```

Usage

```
import psutil

# for checking CPU usage
psutil.cpu_percent()

# For checking disk I/O,
psutil.disk_io_counters()

# For checking the network I/O bandwidth:
psutil.net_io_counters()
```

rsync with python

Use the rsync command in Python

```
import subprocess
src = "<source-path>" # replace <source-path> with the source directory
dest = "<destination-path>" # replace <destination-path> with the destination directory

subprocess.call(["rsync", "-arq", src, dest])
```

Segmentation fault

??????? - ????????????????? C,
C++???

gdb

- `ulimit -c unlimited` : ???? core file ?? unlimited

- `gdb -c <core-file> <program-name>` : ?? core file ???

```
ulimit -c unlimited
gdb -c core example
```

gdb sub-commands

- backtrace : ???????????
- up : ?? backtrace ????????
- list : ???????????
- print : ???????

```
gdb -c core example
....
(gdb) backtrace
....
(gdb) up
...
list
...
print i
...
print argv[0]
...
print argv[1]
```