Programmering af Mobile Robotter

RB1-PMR – Module 7: Debugging, data collection, and visualization

Jes Hundevadt Jepsen | jegj@mmmi.sdu.dk
SDU Drone Center, MMMI, University of Southern Denmark, Odense, Denmark

Agenda

- Recap of last module

- Troubleshooting and Debugging Techniques (continued)
  - Best-practice
    - Basic debugging using Python
    - LED and GPIO for Debugging
    - Data Logging using MicroPython
- Data collection using a Microcontroller
  - Types of data collect
  - End-to-end data collection (design)
- Visualization of data measurements
  - …using Matplotlib

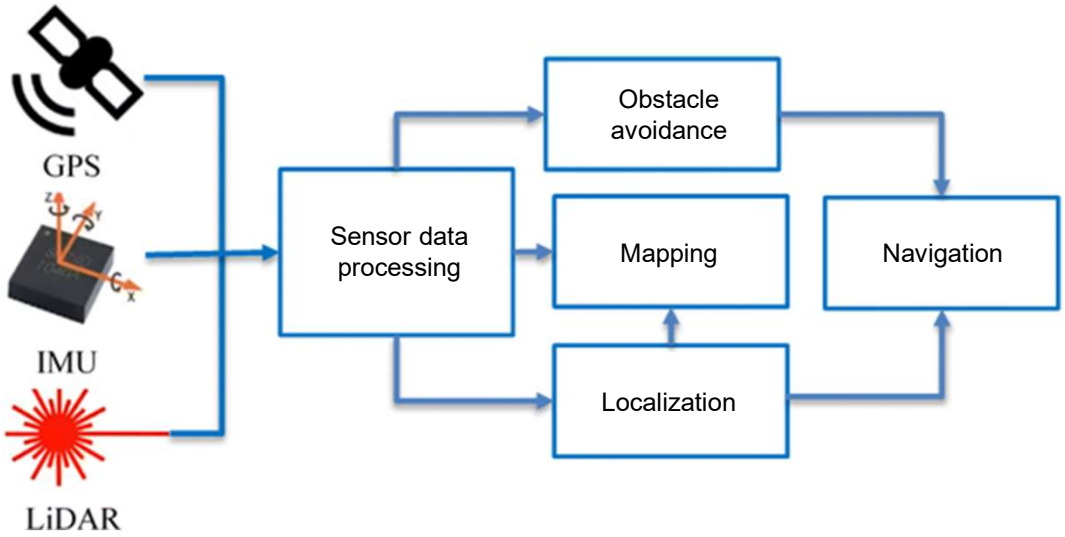
- Portfolio 3: Data collection of sensors
- Assignments
- Fokusgruppeinterview (25/11, 8.15 - 9.15)
  - …af Lea Lukas fra TEK Kommunikation i Sønderborg
  - …omkring der spiller en rolle i uddannelsesvalget?

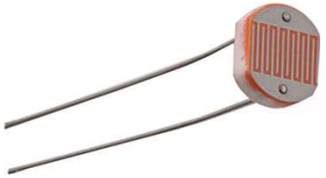
Agenda

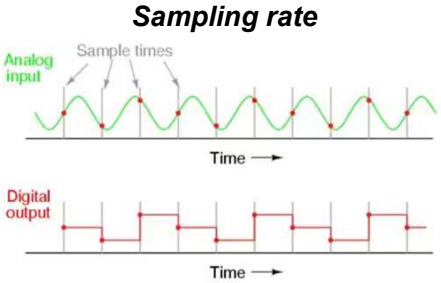
- Sensors
  - Sensor categories
    - Internal versus External
    - Local versus Global
    - Active versus Passive
  - Sensor types
    - Binary sensors
    - Analog sensors
      - Analog-to-Digital Converter (ADC)
    - Digital sensors

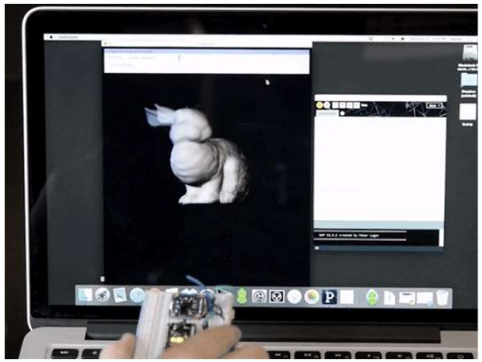
- Robot Behaviors
  - Pseudocode
  - State machines
  - Flowcharts
  - Simple control (using feedback)

- Extra Credit Activity #3
  - Q/A?









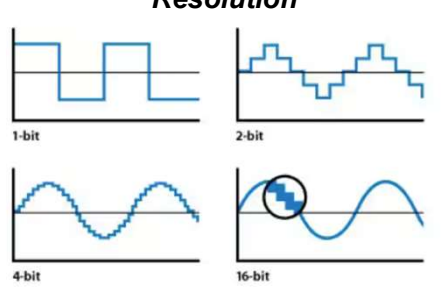


Illustration: Modified from “A robust autonomous navigation and mapping system based on GPS and LiDAR data for unconstraint environment” and Google Images

Troubleshooting and Debugging

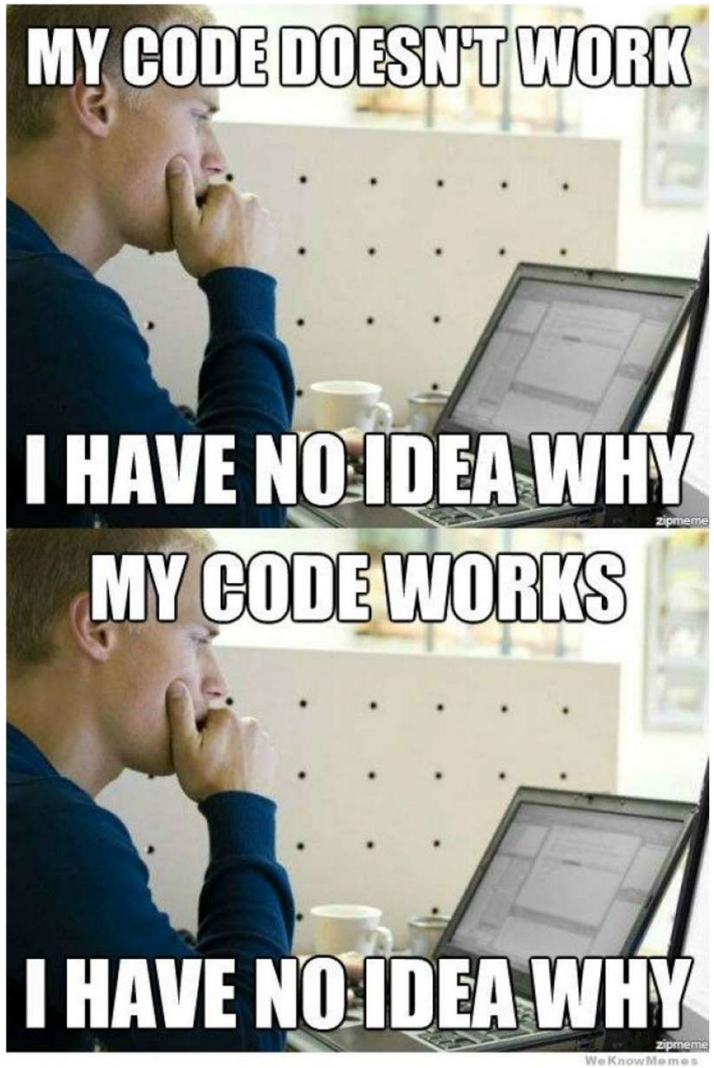
Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Best practise:

"Try again, ask a friend, ask an adult…"

- …when you get an error, look at the:
  - **Error Type**: The type of error (e.g., SyntaxError, TypeError, ValueError).
  - **Traceback**: A list showing the sequence of calls that led to the error.
  - **Error Message**: A description of what caused the error (e.g., "division by zero").



Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Common errors (...from Module 2)**:**

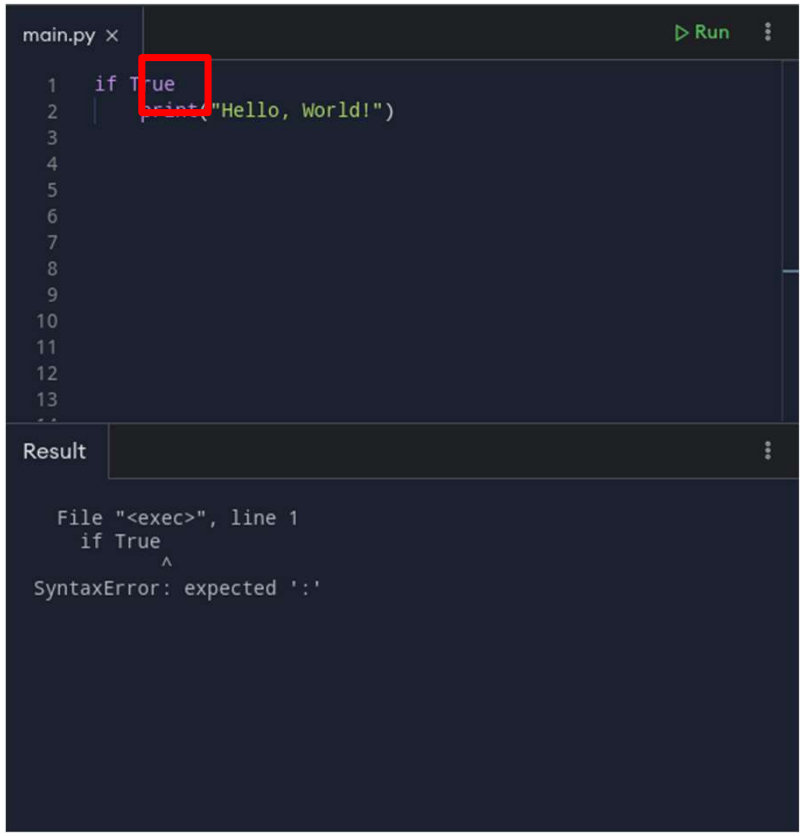
- **Syntax Errors:** Occur when the interpreter encounters code that doesn't conform to the language's syntax rules.
- **Indentation Errors:** Misaligned or inconsistent indentation can lead to errors.
- **Name Errors:** A variable or function is used before it's defined or misspelled.
- **Type Errors:** Occur when an operation or function is applied to an object of inappropriate type (e.g., adding a string to an integer).
- **Index/Key Errors:** When trying to access an index that is out of range in a list or string, or trying to access a dictionary with a key that does not exist.
- **Value Errors:** When a function receives an argument of the correct type but an inappropriate value.

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Common errors (...from Module 2)**:**

- **Syntax Errors:** Occur when the interpreter encounters code that doesn't conform to the language's syntax rules.
- **Indentation Errors:** Misaligned or inconsistent indentation can lead to errors.
- **Name Errors:** A variable or function is used before it's defined or misspelled.
- **Type Errors:** Occur when an operation or function is applied to an object of inappropriate type (e.g., adding a string to an integer).
- **Index/Key Errors:** When trying to access an index that is out of range in a list or string, or trying to access a dictionary with a key that does not exist.
- **Value Errors:** When a function receives an argument of the correct type but an inappropriate value.



Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Common errors (...from Module 2)**:**

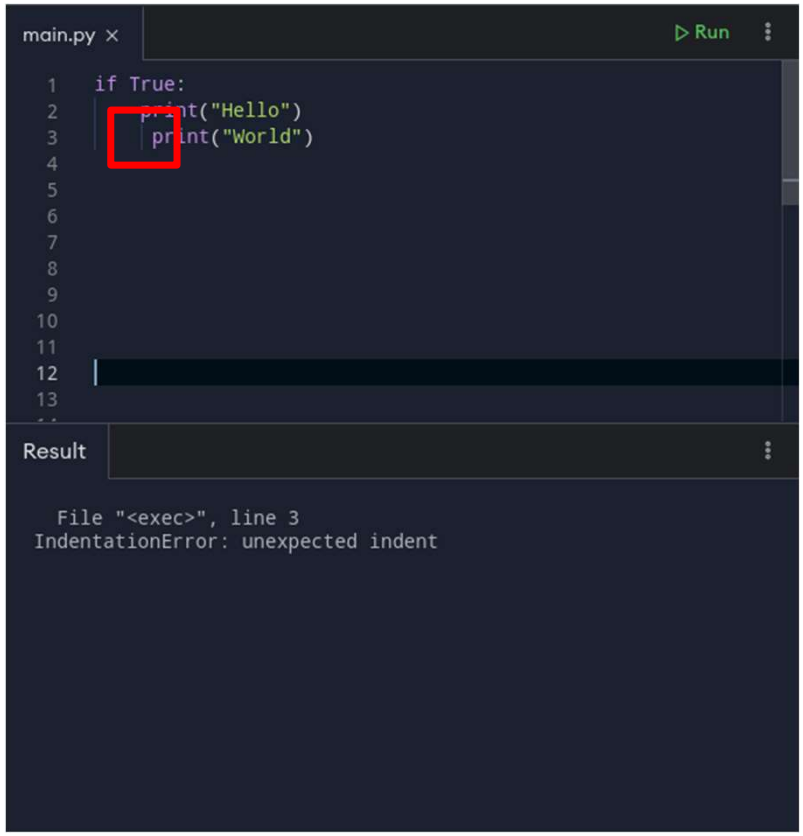
- **Syntax Errors:** Occur when the interpreter encounters code that doesn't conform to the language's syntax rules.
- **Indentation Errors:** Misaligned or inconsistent indentation can lead to errors.
- **Name Errors:** A variable or function is used before it's defined or misspelled.
- **Type Errors:** Occur when an operation or function is applied to an object of inappropriate type (e.g., adding a string to an integer).
- **Index/Key Errors:** When trying to access an index that is out of range in a list or string, or trying to access a dictionary with a key that does not exist.
- **Value Errors:** When a function receives an argument of the correct type but an inappropriate value.

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Common errors (...from Module 2)**:**

- **Syntax Errors:** Occur when the interpreter encounters code that doesn't conform to the language's syntax rules.
- **Indentation Errors:** Misaligned or inconsistent indentation can lead to errors.
- **Name Errors:** A variable or function is used before it's defined or misspelled.
- **Type Errors:** Occur when an operation or function is applied to an object of inappropriate type (e.g., adding a string to an integer).
- **Index/Key Errors:** When trying to access an index that is out of range in a list or string, or trying to access a dictionary with a key that does not exist.
- **Value Errors:** When a function receives an argument of the correct type but an inappropriate value.



Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Common errors (...from Module 2)**:**

- **Syntax Errors:** Occur when the interpreter encounters code that doesn't conform to the language's syntax rules.
- **Indentation Errors:** Misaligned or inconsistent indentation can lead to errors.
- **Name Errors:** A variable or function is used before it's defined or misspelled.
- **Type Errors:** Occur when an operation or function is applied to an object of inappropriate type (e.g., adding a string to an integer).
- **Index/Key Errors:** When trying to access an index that is out of range in a list or string, or trying to access a dictionary with a key that does not exist.
- **Value Errors:** When a function receives an argument of the correct type but an inappropriate value.

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Common errors (...from Module 2)**:**

- **Syntax Errors:** Occur when the interpreter encounters code that doesn't conform to the language's syntax rules.
- **Indentation Errors:** Misaligned or inconsistent indentation can lead to errors.
- **Name Errors:** A variable or function is used before it's defined or misspelled.
- **Type Errors:** Occur when an operation or function is applied to an object of inappropriate type (e.g., adding a string to an integer).
- **Index/Key Errors:** When trying to access an index that is out of range in a list or string, or trying to access a dictionary with a key that does not exist.
- **Value Errors:** When a function receives an argument of the correct type but an inappropriate value.

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Common errors (...from Module 2)**:**

- **Syntax Errors:** Occur when the interpreter encounters code that doesn't conform to the language's syntax rules.
- **Indentation Errors:** Misaligned or inconsistent indentation can lead to errors.
- **Name Errors:** A variable or function is used before it's defined or misspelled.
- **Type Errors:** Occur when an operation or function is applied to an object of inappropriate type (e.g., adding a string to an integer).
- **Index/Key Errors:** When trying to access an index that is out of range in a list or string, or trying to access a dictionary with a key that does not exist.
- **Value Errors:** When a function receives an argument of the correct type but an inappropriate value.

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Common errors (...from Module 2)**:**

- **Syntax Errors:** Occur when the interpreter encounters code that doesn't conform to the language's syntax rules.
- **Indentation Errors:** Misaligned or inconsistent indentation can lead to errors.
- **Name Errors:** A variable or function is used before it's defined or misspelled.
- **Type Errors:** Occur when an operation or function is applied to an object of inappropriate type (e.g., adding a string to an integer).
- **Index/Key Errors:** When trying to access an index that is out of range in a list or string, or trying to access a dictionary with a key that does not exist.
- **Value Errors:** When a function receives an argument of the correct type but an inappropriate value.

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...from Module 2)

- **Print Statements:** Inserting print() statements in your code can help track the flow of execution and check the values of variables at different points.

Example: Using `print()` for debugging

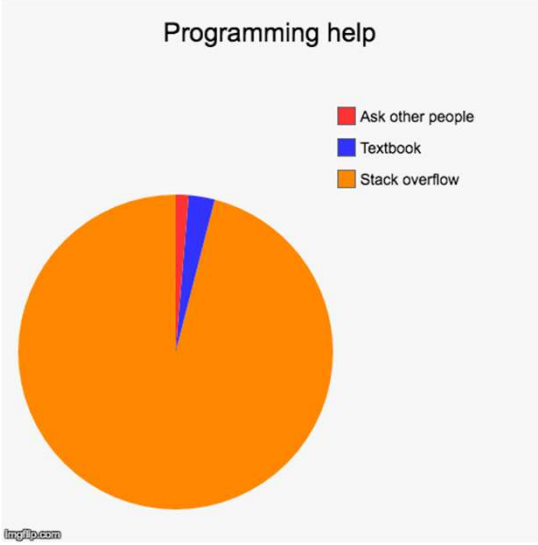
```
print("I'm here")

...

print("Now I'm here...")

...

print("Counter: {}".format(counter) )
print("Counter: {}".format(type(counter)) )
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...from Module 2)

- **Print Statements:** Inserting print() statements in your code can help track the flow of execution and check the values of variables at different points.
- **LMGTFY** (Let Me Google That For You)...
  - Online Q&A / Stack Overflow



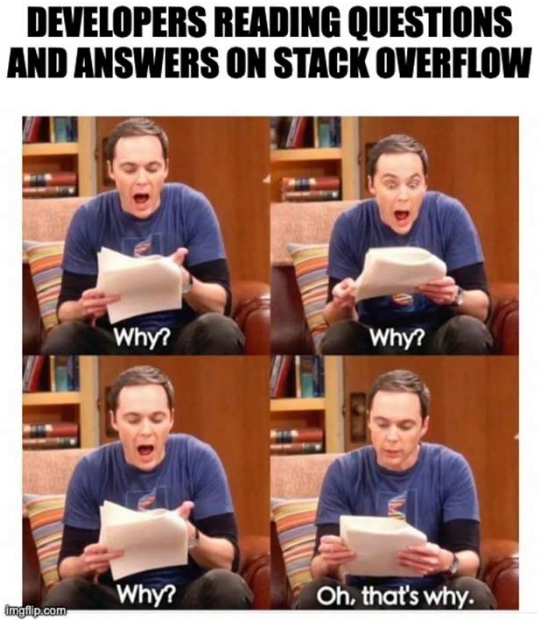


Illustration: From Google images

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...from Module 2)

- **Print Statements:** Inserting print() statements in your code can help track the flow of execution and check the values of variables at different points.
- **LMGTFY** (Let Me Google That For You)...
- **Try-Except Blocks:** Use try-except blocks to catch and handle exceptions, which can help prevent your program from crashing and allow you to manage errors gracefully.

Example: Using `print()` for debugging

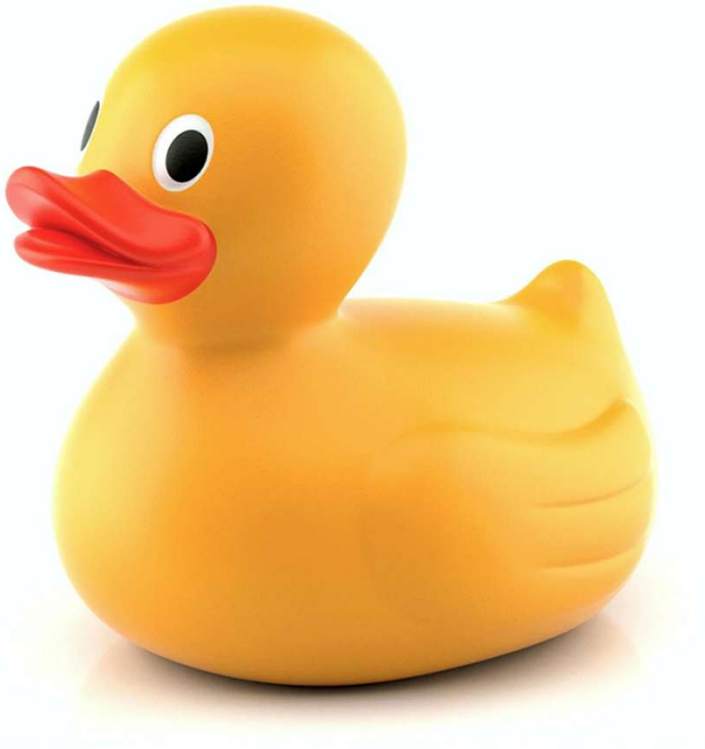
```
print("I'm here")

...

print("Now I'm here...")

...

print("Counter: {}".format(counter) )
print("Counter: {}".format(type(counter)) )
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...from Module 2)

- **Print Statements:** Inserting print() statements in your code can help track the flow of execution and check the values of variables at different points.
- **LMGTFY** (Let Me Google That For You)...
- **Try-Except Blocks:** Use try-except blocks to catch and handle exceptions, which can help prevent your program from crashing and allow you to manage errors gracefully.
- **Rubberdugging / Rubber Duck Debugging:** Explaining your code, often to an inanimate object like a rubber duck, can help clarify your thinking and reveal bugs you might have missed.

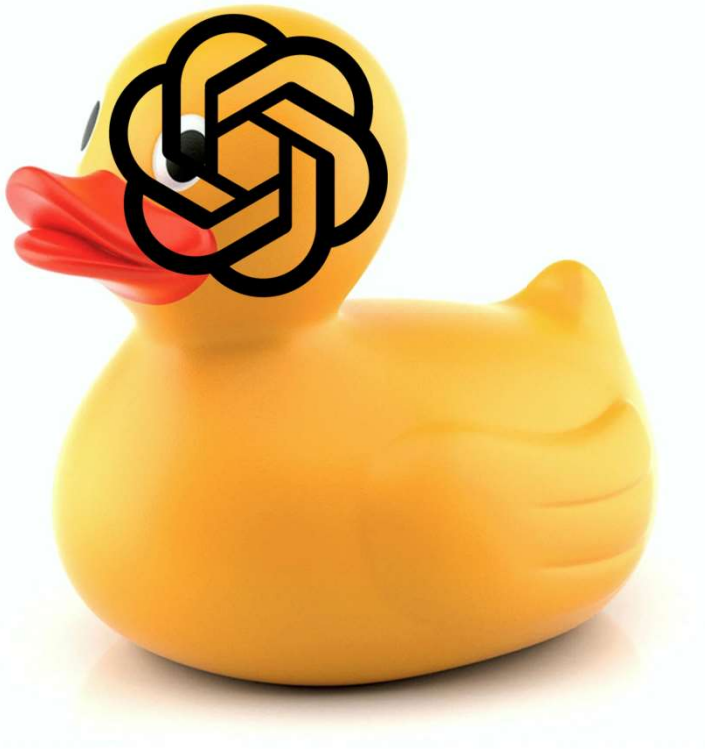


Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...from Module 2)

- **Print Statements:** Inserting print() statements in your code can help track the flow of execution and check the values of variables at different points.
- **LMGTFY** (Let Me Google That For You)...
- **Try-Except Blocks:** Use try-except blocks to catch and handle exceptions, which can help prevent your program from crashing and allow you to manage errors gracefully.
- **Rubberdugging / Rubber Duck Debugging:** Explaining your code, often to an inanimate object like a rubber duck, can help clarify your thinking and reveal bugs you might have missed.
- **Artificial Intelligence…**

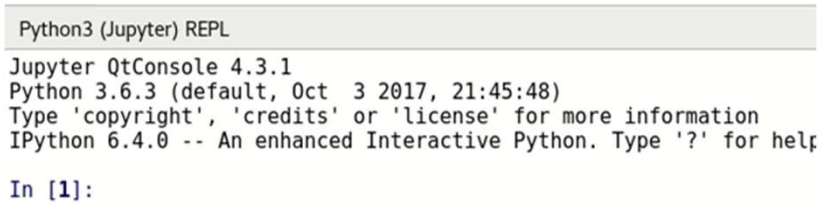


Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...from Module 3)

- **Print Statements:** Inserting print() statements in your code can help track the flow of execution and check the values of variables at different points.
- **LMGTFY** (Let Me Google That For You)...
- **Try-Except Blocks:** Use try-except blocks to catch and handle exceptions, which can help prevent your program from crashing and allow you to manage errors gracefully.
- **Rubberdugging / Rubber Duck Debugging:** Explaining your code, often to an inanimate object like a rubber duck, can help clarify your thinking and reveal bugs you might have missed.
- **Artificial Intelligence…**
- **REPL (Read-Eval-Print-Loop):** interactive shell that allows you to run Python commands in real-time directly on the Pico.



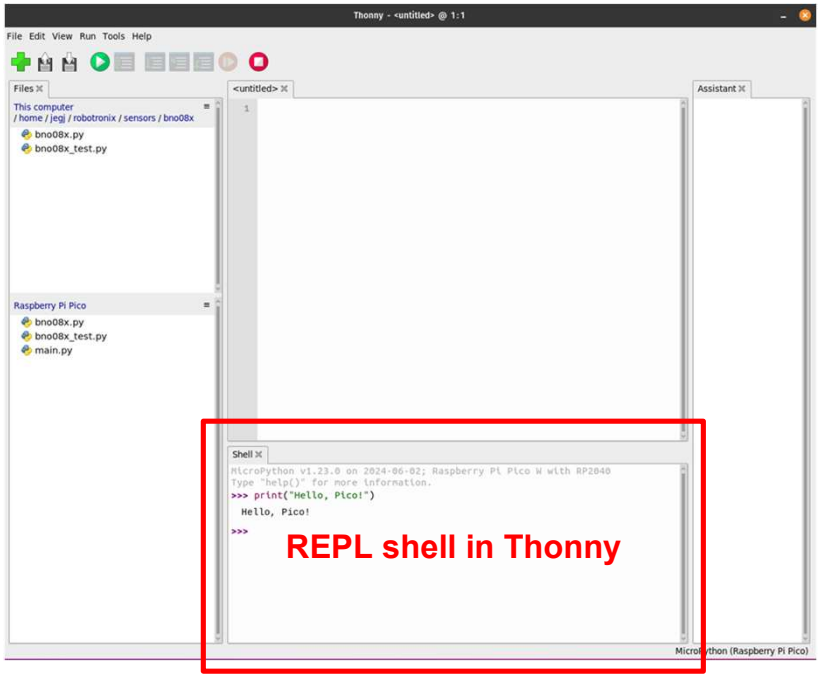


Illustration: https://codewith.mu/en/tutorials/1.2/repl

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

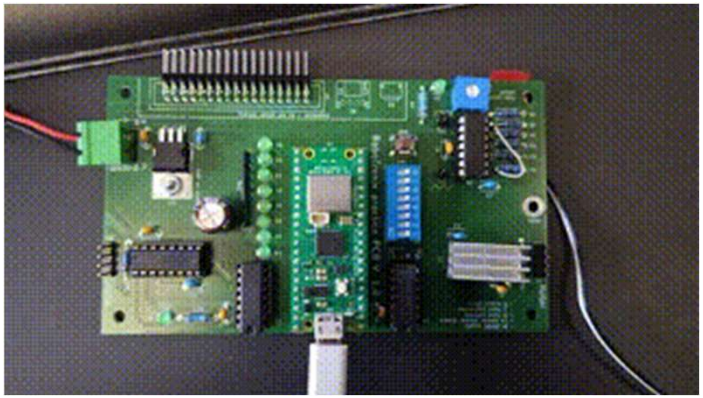
- **LED and GPIO for Debugging**

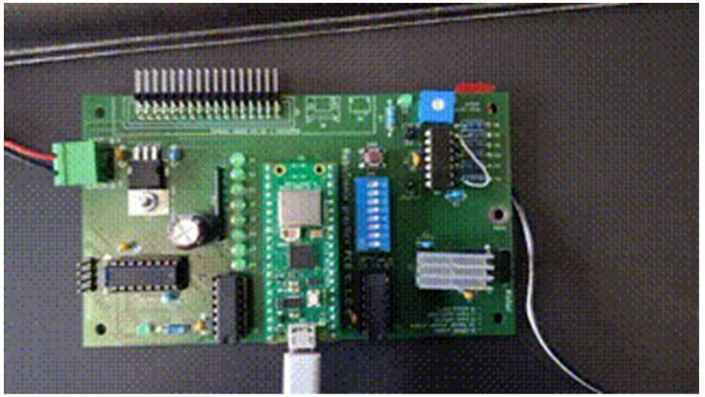
Troubleshooting and Debugging

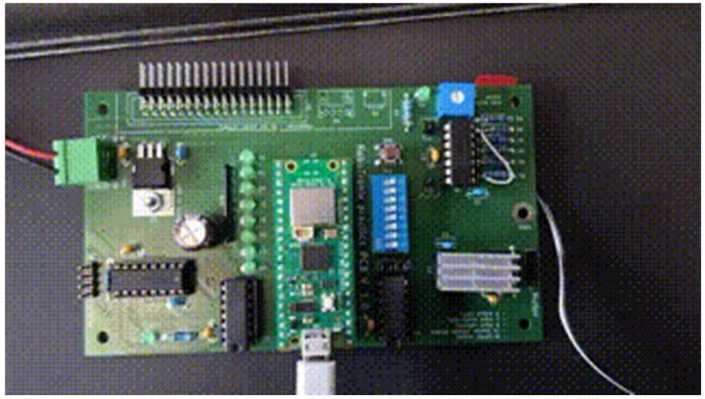
"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging**
  - **Status LEDs:** use the built-in LED or external GPIO pins to indicate certain states in your code, eg. turn on the LED(s) or a blink in a certain pattern when either reaching a specific point in your code or a specific state







Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging**
  - **Status LEDs:** use the built-in LED or external GPIO pins to indicate certain states in your code, eg. turn on the LED(s) or a blink in a certain pattern when either reaching a specific point in your code or a specific state

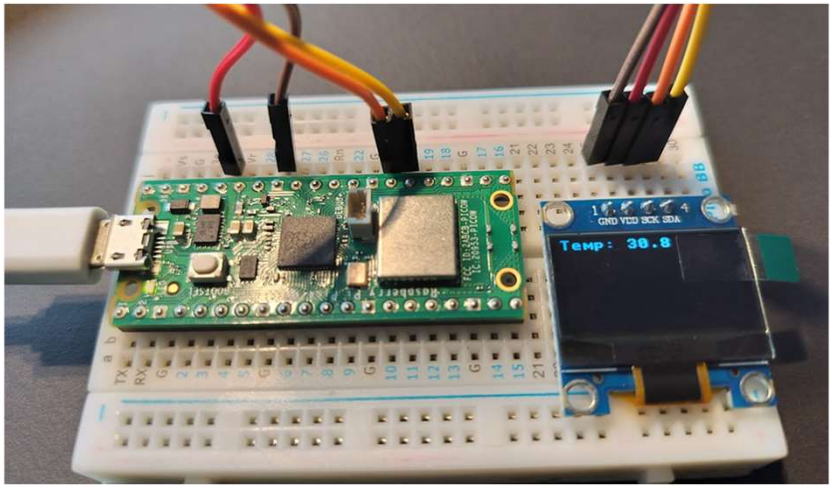
  - **Signal monitoring:** Use an oscilloscope or logic analyzer to monitor the signals on GPIO pins. Useful for:
    - Validating PWM signals
    - debugging communication protocols (I2C, SPI, UART).
      - …more about that in Module 9

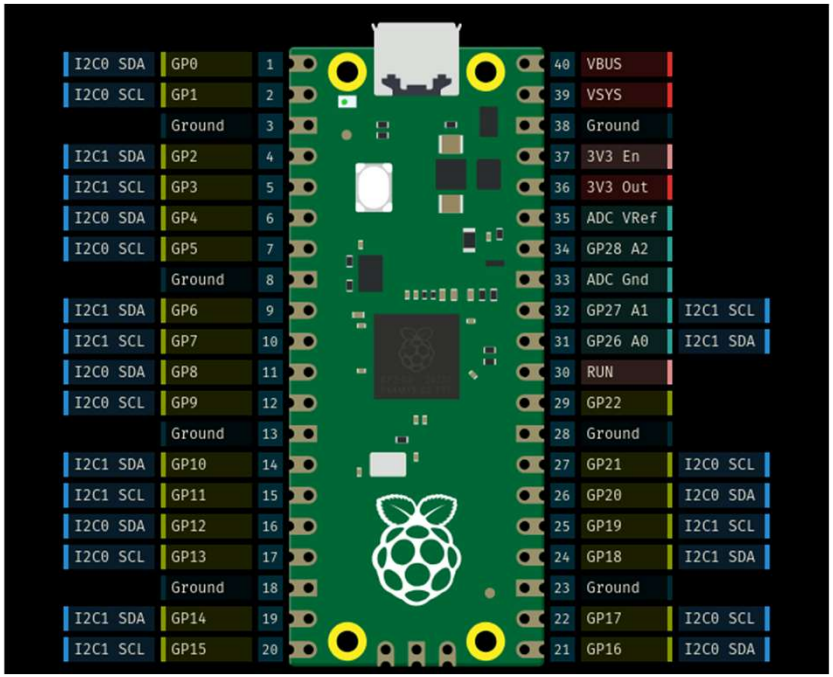
Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging**
  - **Status LEDs:** use the built-in LED or external GPIO pins to indicate certain states in your code, eg. turn on the LED(s) or a blink in a certain pattern when either reaching a specific point in your code or a specific state
  - **Signal monitoring:** Use an oscilloscope or logic analyzer to monitor the signals on GPIO pins. Useful for:
    - Validating PWM signals
    - debugging communication protocols (I2C, SPI, UART).
      - …more about that in Module 9
  - **Printing on small/portable "external screens"**, eg. OLEDs
    - Real-time monitoring of the output
    - …using I2C, SPI, or UART
    - …same procedure as with digital sensors…



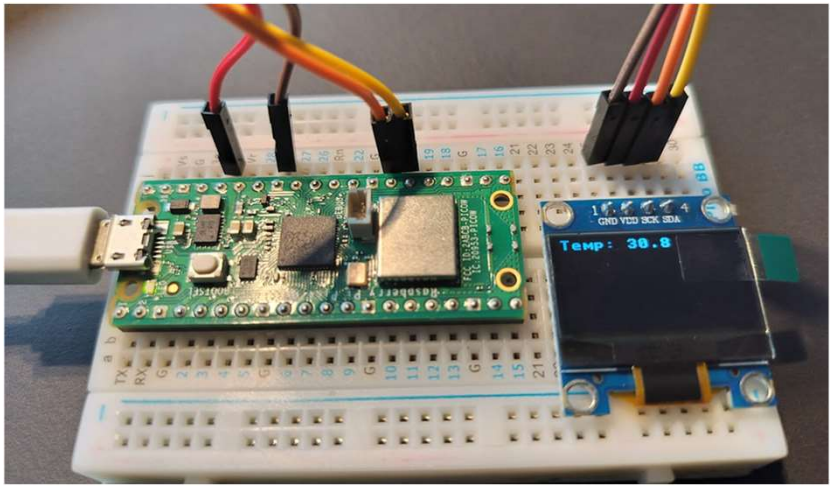


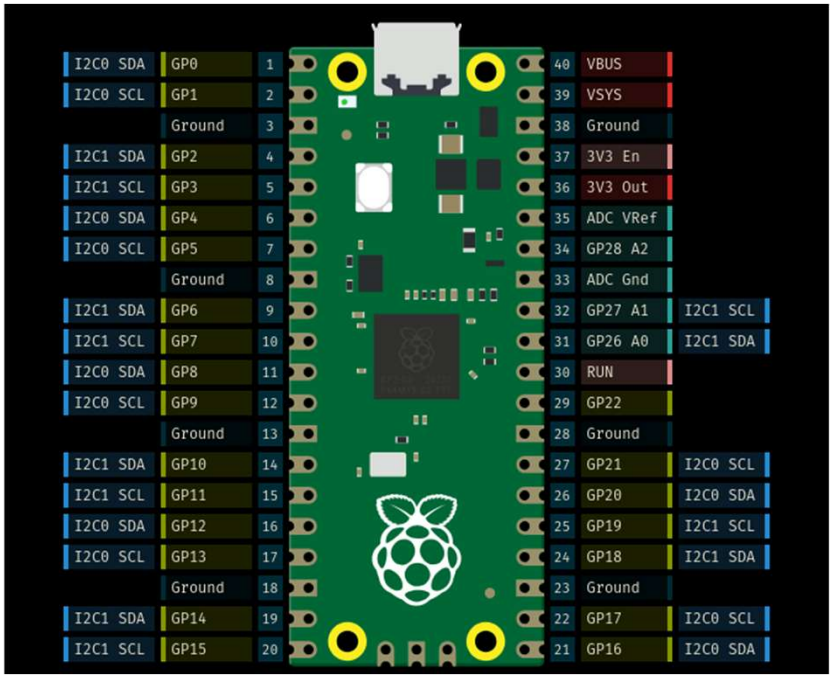
Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging**
  - **Status LEDs:** use the built-in LED or external GPIO pins to indicate certain states in your code, eg. turn on the LED(s) or a blink in a certain pattern when either reaching a specific point in your code or a specific state
  - **Signal monitoring:** Use an oscilloscope or logic analyzer to monitor the signals on GPIO pins. Useful for:
    - Validating PWM signals
    - debugging communication protocols (I2C, SPI, UART).
      - …more about that in Module 9
  - **Printing on small/portable "external screens"**, eg. OLEDs





SDU

Demo?

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- LED and GPIO for Debugging
- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
    - **Example:** Log each button press
        - …by writing to a local file.
        - Important information?

Example: Button press using Interrupts

```
from machine import Pin
import time
import uos

button = Pin("GP27", Pin.IN, Pin.PULL_UP)

def log_message(message):
    with open("log.txt", "a") as log_file:
        log_file.write(message + "\n")

while True:
  first = button.value()
  time.sleep(0.01)
  second = button.value()
  if first and not second:
      print('Button pressed!') # Print and write to log when pressed
      log_message("Button pressed")
  elif not first and second:
      print('Button released!') # ... and only print when it is
released
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …
- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
  - **Example:** Log each button press
    - Timestamp, eg. using `time.time()`
      - …when a button is pressed

Example: Button press using Interrupts

```
from machine import Pin
import time
import uos

button = Pin("GP27", Pin.IN, Pin.PULL_UP)

def log_message(message):
    with open("log.txt", "a") as log_file:
        log_file.write(message + "\n")

while True:
  first = button.value()
  time.sleep(0.01)
  second = button.value()
  if first and not second:
      print('Button pressed!') # Print and write to log when pressed
      log_message("Button pressed at {}".format(time.time()))
  elif not first and second:
      print('Button released!') # ... and only print when it is
released
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …
- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
  - **Example:** Log each button press
    - Timestamp, eg. using `time.time()`
      - …when a button is pressed…
      - …and when the system has started up.
    - Others?

Example: Button press using Interrupts

```
from machine import Pin
import time
import uos

button = Pin("GP27", Pin.IN, Pin.PULL_UP)

def log_message(message):
    with open("log.txt", "a") as log_file:
        log_file.write(message + "\n")

log_message("Log started at {}".format(time.time()))
while True:
  first = button.value()
  time.sleep(0.01)
  second = button.value()
  if first and not second:
      print('Button pressed!') # Print and write to log when pressed
      log_message("Button pressed at {}".format(time.time()))
  elif not first and second:
      print('Button released!') # ... and only print when it is
released
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …

- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.

    - **Example:** Log each button press
        - Timestamp, eg. using `time.time()`
            - …when a button is pressed…
            - …and when the system has started up.

        - Others:
            - Number of total button presses?
            - Time since last button press?

Example: Button press using Interrupts

```
from machine import Pin
import time
import uos

button = Pin("GP27", Pin.IN, Pin.PULL_UP)

def log_message(message):
   with open("log.txt", "a") as log_file:
       log_file.write(message + "\n")

log_message("Log started at {}".format(time.time()))
while True:
  first = button.value()
  time.sleep(0.01)
  second = button.value()
  if first and not second:
      print('Button pressed!') # Print and write to log when pressed
      log_message("Button pressed at {}".format(time.time()))
  elif not first and second:
      print('Button released!') # ... and only print when it is
released
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …
- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
    - **Example:** Output from the logging of button presses
        - Timestamp: `1726584534 ?`

Example: Log file (log.txt) after three boot-ups and several button presses

```
Log started at 1726584529
Button pressed at 1726584534
Button pressed at 1726584541
Button pressed at 1726584545
Button pressed at 1726584545
Button pressed at 1726584546
Button pressed at 1726584546
Button pressed at 1726584550
Log started at 1726584564
Button pressed at 1726584573
Button pressed at 1726584579
Button pressed at 1726584589
Log started at 1609459201
Button pressed at 1609459204
Button pressed at 1609459209
Button pressed at 1609459209
Button pressed at 1609459219
Button pressed at 1609459225
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …

- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.

    - **Example:** Output from the logging of button presses
        - Timestamp: `1726584534 ?`

        - **Epoch:** refers to the starting point used to measure time.
            - Unix Epoch: January 1, 1970, 00:00:00
            - **…seconds since this point.**

        Convert for being human-readable, eg. https://www.epochconverter.com/

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

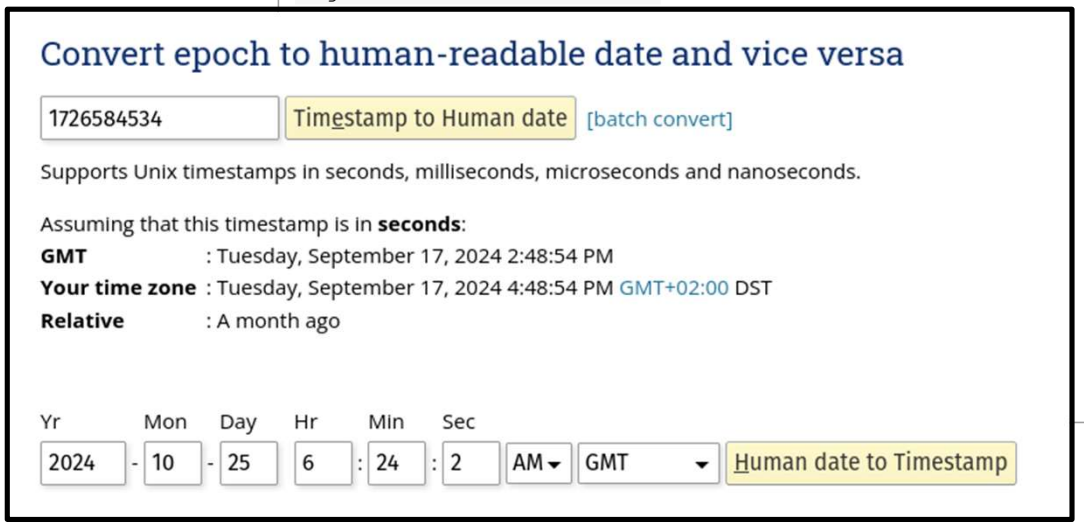
Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …
- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
  - **Example:** Output from the logging of button presses
    - So `1726584534` is converted to

Tuesday, September 17, 2024 2:48:54 PM

Example: Log file (log.txt) after three boot-ups and several button presses

```
Log started at 1726584529
Button pressed at 1726584534
Button pressed at 1726584541
Button pressed at 1726584545
Button pressed at 1726584545
Button pressed at 1726584546
Button pressed at 1726584546
Button pressed at 1726584550
Log started at 1726584564
```



Source: https://www.epochconverter.com/

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …

- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.

    - **Example:** Output from the logging of button presses
        - So `1726584534` is converted to

        Tuesday, September 17, 2024 2:48:54 PM

        - Why `1609459201?`

        Friday, January 1, 2021 12:00:01 AM

        (RP2040 release date?)

Example: Log file (log.txt) after three boot-ups and several button presses

```
Log started at 1726584529
Button pressed at 1726584534
Button pressed at 1726584541
Button pressed at 1726584545
Button pressed at 1726584545
Button pressed at 1726584546
Button pressed at 1726584546
Button pressed at 1726584550
Log started at 1726584564
Button pressed at 1726584573
Button pressed at 1726584579
Button pressed at 1726584589
Log started at 1609459201
Button pressed at 1609459204
Button pressed at 1609459209
Button pressed at 1609459209
Button pressed at 1609459219
Button pressed at 1609459225
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …

- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
  - **Logged** when testing using Thonny (time sync)

  - **Logged** when no laptop is connected (no time sync)
    - Pico doesn't have a built-in Real-Time Clock (RTC)
      - …meaning it won't keep time when powered off.

    - …always starts at Friday, January 1, 2021 12:00:01 AM

Example: Log file (log.txt) after three boot-ups and several button presses

```
Log started at 1726584529
Button pressed at 1726584534
Button pressed at 1726584541
Button pressed at 1726584545
Button pressed at 1726584545
Button pressed at 1726584546
Button pressed at 1726584546
Button pressed at 1726584550
Log started at 1726584564
Button pressed at 1726584573
Button pressed at 1726584579
Button pressed at 1726584589
Log started at 1609459201
Button pressed at 1609459204
Button pressed at 1609459209
Button pressed at 1609459209
Button pressed at 1609459219
Button pressed at 1609459225
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …

- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
  - **Logged** when testing using Thonny (time sync)

  - **Logged** when no laptop is connected (no time sync)
    - Pico doesn't have a built-in Real-Time Clock (RTC)
      - …meaning it won't keep time when powered off.

    - …always starts at Friday, January 1, 2021 12:00:01 AM
      - …multiple logging with the same timestamps…

How to avoid this issue?

Example: Log file (log.txt) after three boot-ups and several button presses

```
Log started at 1726584529
Button pressed at 1726584534
Button pressed at 1726584541
Button pressed at 1726584545
Button pressed at 1726584545
Button pressed at 1726584546
Button pressed at 1726584546
Button pressed at 1726584550
Log started at 1726584564
Button pressed at 1726584573
Button pressed at 1726584579
Button pressed at 1726584589
Log started at 1609459201
Button pressed at 1609459204
Button pressed at 1609459209
Button pressed at 1609459209
Button pressed at 1609459219
Button pressed at 1609459225
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …

- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
  - (Beyond scope) Use a Real-Time Clock (RTC)
    - Additional hardware, maintains time when the power is off.

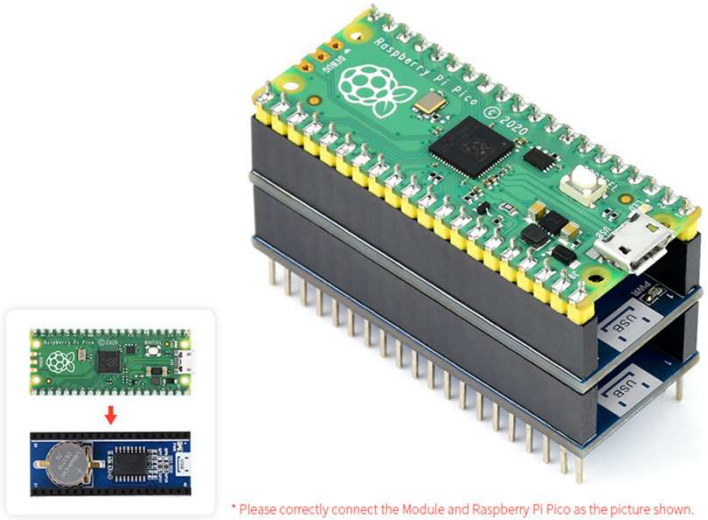


Illustration: https://www.waveshare.com/pico-rtc-ds3231.htm

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …

- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
  - (Beyond scope) Use a Real-Time Clock (RTC)
    - Additional hardware, maintains time when the power is off.
  - (Beyond scope) Use an NTP server (Pico W only).
    - If connected to a Wi-Fi network with internet access, get the current time from an Network Time Protocol (NTP) server, and set the internal clock

Example: Use Wi-Fi to get the current time

```
import network
import time
import ntptime

# Configure your Wi-Fi credentials
ssid = 'your-SSID'
password = 'your-password'

# Initialize the Wi-Fi interface
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)

# Wait until the Wi-Fi is connected
while not wlan.isconnected():
    time.sleep(1)

print("Connected to Wi-Fi")

# Synchronize the time with an NTP server
try:
    ntptime.settime()  # This sets the internal clock to the current UTC
time
    print("Time synchronized with NTP server")
except:
    print("Failed to synchronize time")

# Get the current time and print it
current_time = time.localtime()
print("Current time:", current_time)
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …

- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
  - Overwrite the existing log-file
    - When you want to append to a log file, you open the file in append mode (`'a'`)
      - You may run into memory-related issues, when there is to many log-files…
    - If you want to overwrite the log file each time you write to it, you open the file in write mode (`'w'`)
      - you may run into memory-related issues, if the log-file gets to large…

Example: Append to a log-file

```
# Example: Appending to a log file
log_filename = "logfile.txt"

# Open the file in append mode
with open(log_filename, 'a') as log_file:
    # Write the log entry
    log_file.write("This is a new log entry.\n")

print(f"Log entry appended to {log_filename}")
```

Example: Overwrite a log-file

```
# Example: Overwriting a log file
log_filename = "logfile.txt"

# Open the file in write mode
with open(log_filename, 'w') as log_file:
    # Write the log entry
    log_file.write("This is a new log entry.\n")

print(f"Log entry overwritten in {log_filename}")
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …

- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
  - Overwrite the existing log-file
    - When you want to append to a log file, you open the file in append mode (`'a'`)
    - If you want to overwrite the log file each time you write to it, you open the file in write mode (`'w'`)
  - Create multiple log-files
    - (if time sync) Timestamp each log-file

Example: Append to a log-file

```
import time

# Get the current time and format it as a string
current_time = time.localtime()
timestamp = "{:04d}{:02d}{:02d}_{:02d}{:02d}{:02d}".format(
    current_time[0],  # Year
    current_time[1],  # Month
    current_time[2],  # Day
    current_time[3],  # Hour
    current_time[4],  # Minute
    current_time[5]   # Second
)

# Create the log filename with the current date and time stamp
log_filename = "logs/logfile_{}.txt".format(timestamp)

# Open the file in write mode
with open(log_filename, 'w') as log_file:
    # Write the log entry
    log_file.write("This is a new log entry.\n")

print(f"Log entry: {log_filename}")
```

```
Output:
Log entry: logs/logfile_20240920_082358.txt
Log entry: logs/logfile_20240920_082518.txt
...
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …
- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
  - Overwrite the existing log-file
    - When you want to append to a log file, you open the file in append mode (`'a'`)
    - If you want to overwrite the log file each time you write to it, you open the file in write mode (`'w'`)
  - Create multiple log-files
    - (if time sync) Timestamp each log-file
    - Ensure that you creates a new log-file, without overwriting an existing

Example: Append to a log-file

```python
import os

def count_txt_files():
    # Get a list of all files in the current directory
    files = os.listdir()

    # Filter out only .txt files
    txt_files = [f for f in files if f.endswith(".txt")]

    # Return the count of .txt files
    return len(txt_files)

# Count the number of .txt files
txt_file_count = count_txt_files()

# Create a new log file with an index
log_filename = "logs/logfile_{}.txt".format(txt_file_count+1)

# Open the file in write mode
with open(log_filename, 'w') as log_file:
  # Write the log entry
  log_file.write("This is a new log entry.\n")

print(f"Log entry: {log_filename}")
```

```
Output:
Log entry: logs/logfile_1.txt
Log entry: logs/logfile_2.txt
...
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **LED and GPIO for Debugging:** use the built-in LED or …

- **Logging:** implement a simple logging system by appending messages or variable states to a file stored in the Pico's filesystem.
  - Circular/Ring logging
    - …for an more efficient way and prevent memory overflow.

    - Typically, we are only interested in the last log messages (when debugging)

      - Overwriting the oldest logs with new ones.
        - "Fixed"-sized memory usage, eg. only storing the last five logs

Example: Append to a log-file

```
import os

def count_txt_files():
    # Get a list of all files in the current directory
    files = os.listdir()

    # Filter out only .txt files
    txt_files = [f for f in files if f.endswith(".txt")]

    # Return the count of .txt files
    return len(txt_files)

# Count the number of .txt files
txt_file_count = count_txt_files()

# Create a new log file with an index
log_filename = "logs/logfile_{}.txt".format(txt_file_count+1)

# Open the file in write mode
with open(log_filename, 'w') as log_file:
  # Write the log entry
  log_file.write("This is a new log entry.\n")

print(f"Log entry: {log_filename}")
```

```
Output:
Log entry: logs/logfile_1.txt
Log entry: logs/logfile_2.txt
...
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **Monitoring Memory Usage:** using the garbage collection (gc): https://docs.micropython.org/en/latest/library/gc.html
  - … keeping track of objects that a program creates and remove (or "collect") those that are no longer in use, freeing up memory and preventing memory leaks. So basically:
    - **Tracking Objects:** 1) monitors the objects that the program creates and 2) checks which ones are still reachable or referenced by the program.
    - **Identifying Unused Objects:** identifying the objects that are no longer being used by the program (not reachable or referenced).
    - **Collecting Garbage:** Once identified, then removed ("collected"), making the memory available for new

Illustration: https://www.cegal.com/en/tech-blog/jvm-garbage-collector

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **Monitoring Memory Usage:** using the garbage collection (gc): https://docs.micropython.org/en/latest/library/gc.html
  - **MicroPython** uses automatic garbage collection to manage memory.
    - …meaning it automatically frees up memory that is no longer referenced by any part of your program.

Illustration: https://www.cegal.com/en/tech-blog/jvm-garbage-collector

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **Monitoring Memory Usage:** using the garbage collection (gc): https://docs.micropython.org/en/latest/library/gc.html
  - **MicroPython** uses automatic garbage collection to manage memory.
    - …meaning it automatically frees up memory that is no longer referenced by any part of your program.
  - …however, in some cases…
    - …it can be beneficial to manually trigger garbage collection to ensure memory is released promptly
    - …especially after creating and destroying many objects or large data structures.

Illustration: https://www.cegal.com/en/tech-blog/jvm-garbage-collector

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **Monitoring Memory Usage:** using the garbage collection (gc): https://docs.micropython.org/en/latest/library/gc.html

- Best Practices for using the gc:
  - **gc.mem_free()**: Return the number of bytes of heap RAM that are allocated by Python code.
    - …useful for checking how much memory is left for your program to use
  - **gc.mem_alloc()**: Returns the number of bytes currently allocated in the heap.
    - …useful for understanding how much memory your program is using at any given time.
  - **gc.collect()**: Forces a garbage collection cycle.
    - …useful for freeing up memory that is no longer in use.

Example: Append to a log-file

```
import gc  # Import the garbage collection module

# Check the amount of free memory available in the heap
free_memory = gc.mem_free()
print("Free memory:", free_memory, "bytes")

# Check the amount of memory currently allocated in the heap
allocated_memory = gc.mem_alloc()
print("Allocated memory:", allocated_memory, "bytes")

# Force a garbage collection cycle to free up memory that is no longer in use
gc.collect()
print("Memory after GC:", gc.mem_free(), "bytes")
```

Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- **Monitoring Memory Usage:** using the garbage collection (`gc`): https://docs.micropython.org/en/latest/library/gc.html

- Best Practices for using the `gc` (examples):
  - Call `gc.collect()` after Memory-Intensive Operations
  - Periodic call `gc.collect()`, especially during long-running programs that processes a lot of data.
  - Use `gc.mem_free()` and `gc.mem_alloc()` before and after operations for identifying memory leaks

Example: Append to a log-file

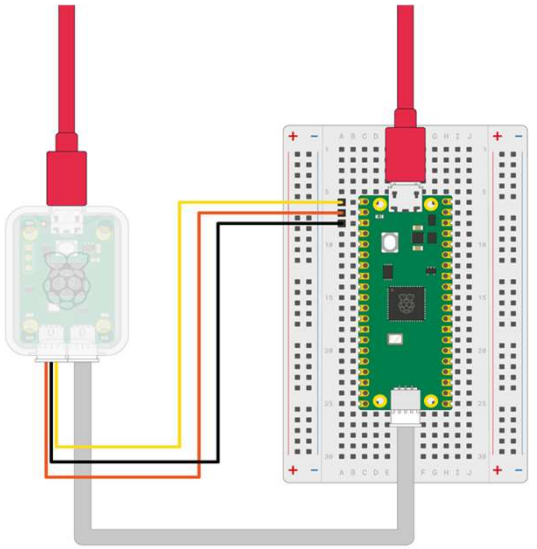
```
import gc

def process_data():
    large_list = [i for i in range(10_000)]

    # Process the data...

    # After processing, the large_list is no longer needed
    del large_list  # Delete the reference (Optional)
    gc.collect()  # Free up memory

process_data()
print("Memory after processing:", gc.mem_free(), "bytes")
```

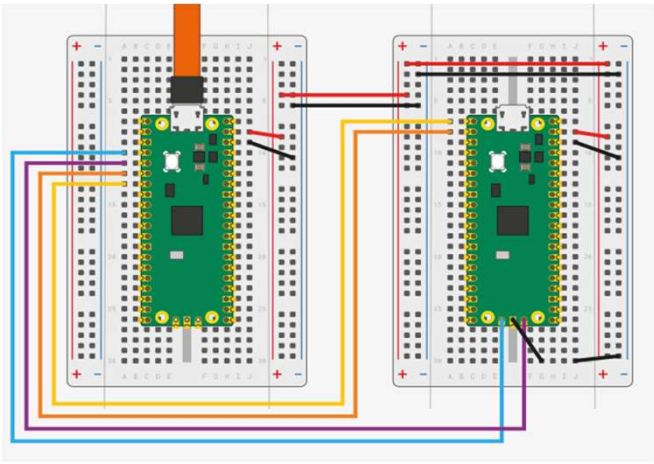
Troubleshooting and Debugging

"...identifying, diagnosing, and resolving errors or bugs that arise when writing or running the code."

Debugging Techniques (...continued)

- (Beyond scope) **Debugging with an External Debugger:** If you need a more advanced debugging tool, you can connect an external hardware debugger.
  - Picoprobe: https://github.com/raspberrypi/debugprobe
  - Using
    - A Raspberry Pi Debug Probe kit for Pico, or
    - Another Raspberry Pi Pico





Illustrations: https://www.raspberrypi.com/documentation/microcontrollers/debug-probe.html
https://mcuoneclipse.com/2022/09/17/picoprobe-using-the-raspberry-pi-pico-as-debug-probe/

Data collection

Data collection

*"...a systematic process of **gathering, measuring, and analyzing information** from various sources to answer research questions, test hypotheses, or evaluate outcomes."*

Illustration: https://www.linkedin.com/pulse/data-everywhere-rob-dabbs

Data collection

"...a systematic process of gathering, measuring, and analyzing information from various sources to answer research questions, test hypotheses, or evaluate outcomes."

Types of data collection

Data collection

"...a systematic process of gathering, measuring, and analyzing information from various sources to answer research questions, test hypotheses, or evaluate outcomes."

Types of data collection

- **Qualitative data** (e.g., interviews, observations, open-ended survey responses).
- **Quantitative data** (e.g., numerical measurements, structured surveys, statistics).

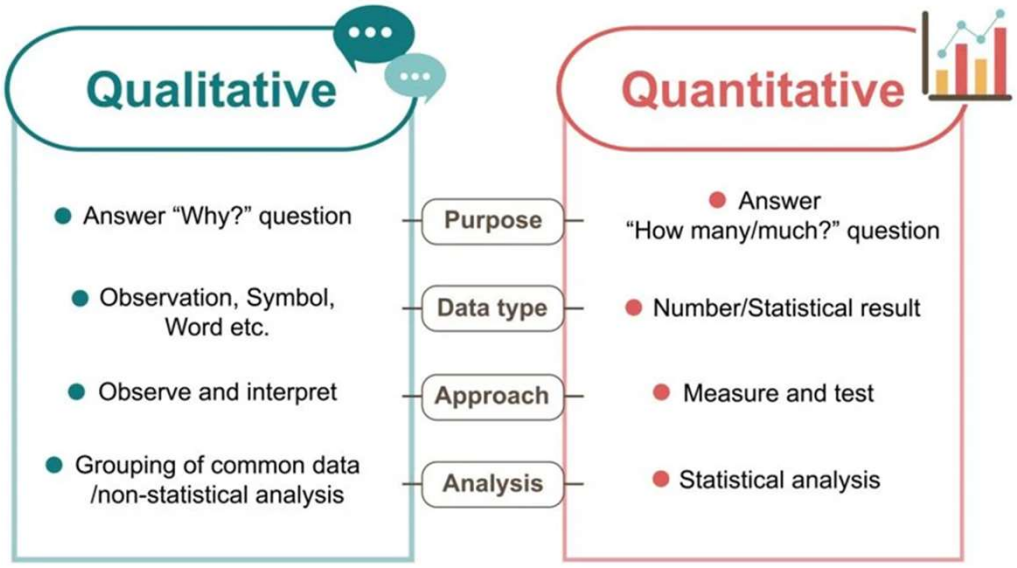


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

Data collection

"...a systematic process of gathering, measuring, and analyzing information from various sources to answer research questions, test hypotheses, or evaluate outcomes."

Types of data collection

- **Qualitative data** (e.g., interviews, observations, open-ended survey responses).
- **Quantitative data** (e.g., numerical measurements, structured surveys, statistics).
- **Primary data** (collected directly from original sources).
- **Secondary data** (collected from existing sources like reports, studies, and databases).

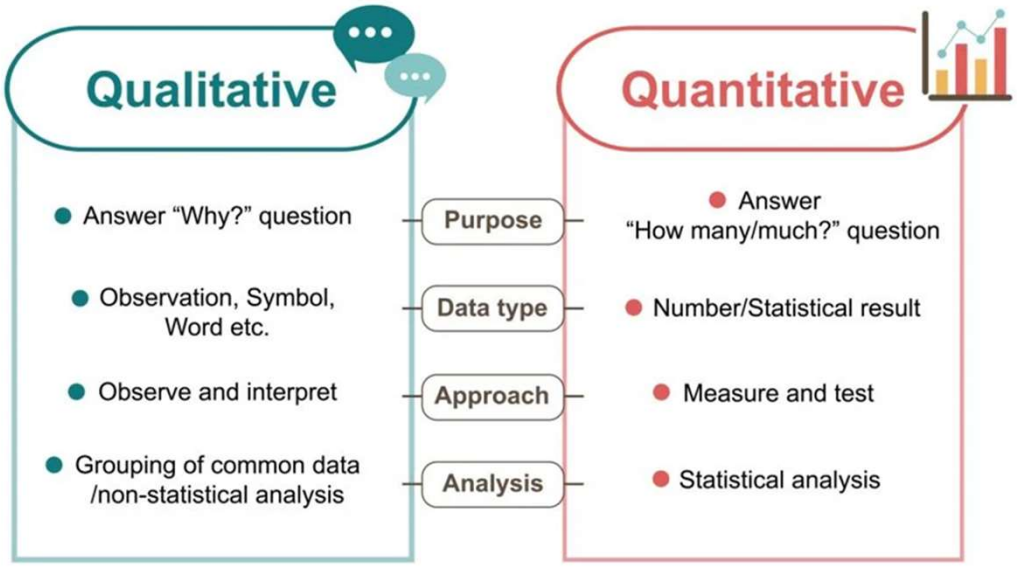


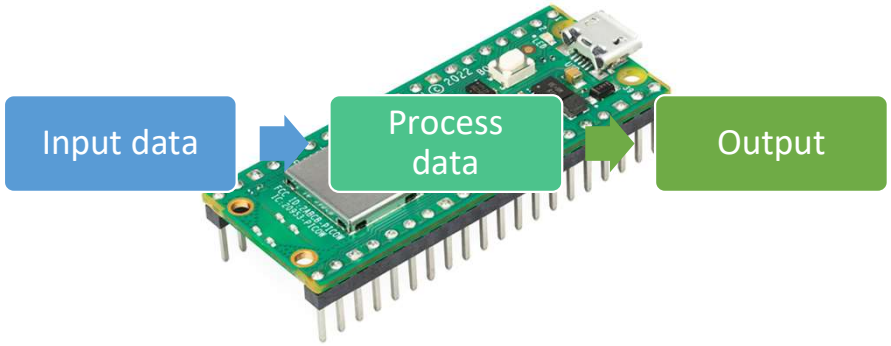
Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

Data collection

"...a systematic process of gathering, measuring, and analyzing information from various sources to answer research questions, test hypotheses, or evaluate outcomes."

Types of data collection

- **Qualitative data** (e.g., interviews, observations, open-ended survey responses).
- **Quantitative data** (e.g., numerical measurements, structured surveys, statistics).
- **Primary data** (collected directly from original sources).
- **Secondary data** (collected from existing sources like reports, studies, and databases).



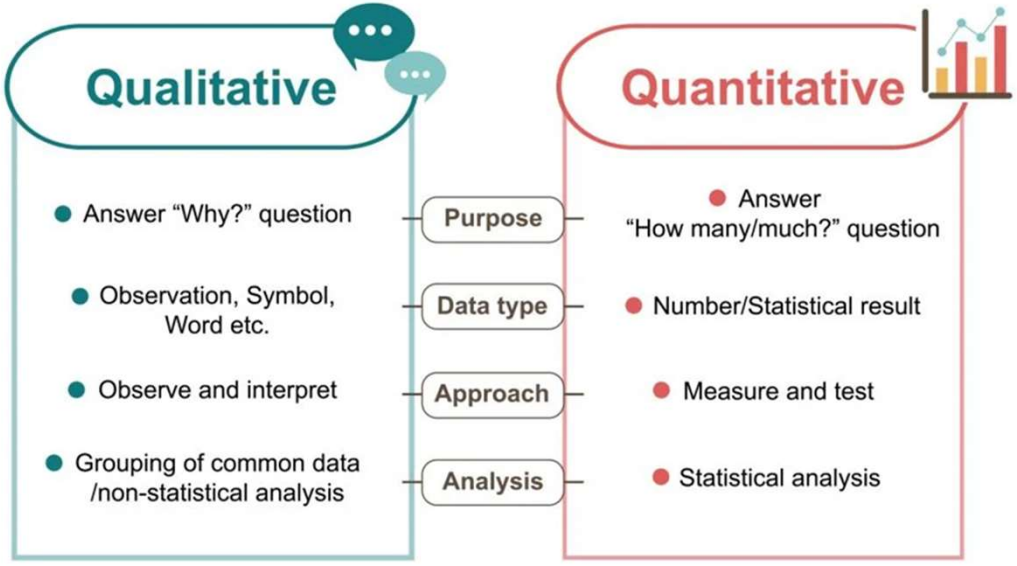


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

Data collection

"...a systematic process of gathering, measuring, and analyzing information from various sources to answer research questions, test hypotheses, or evaluate outcomes."

Why is data collection important?

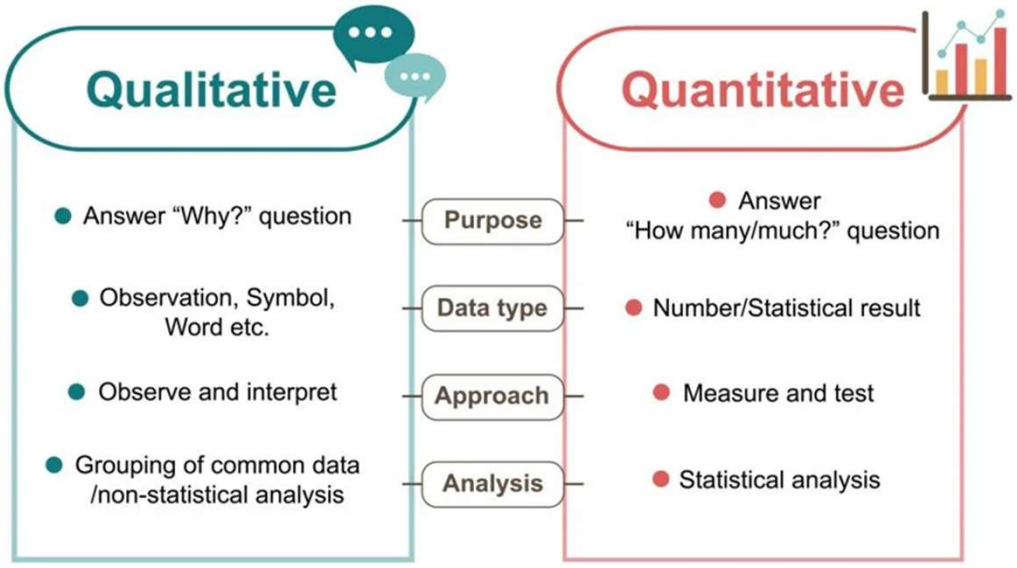


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

Data collection

"...a systematic process of gathering, measuring, and analyzing information from various sources to answer research questions, test hypotheses, or evaluate outcomes."

Why?

- **Insight and decision making:** Proper data collection leads to evidence-based conclusions that can be generalized or applied in applications. **(debugging)**

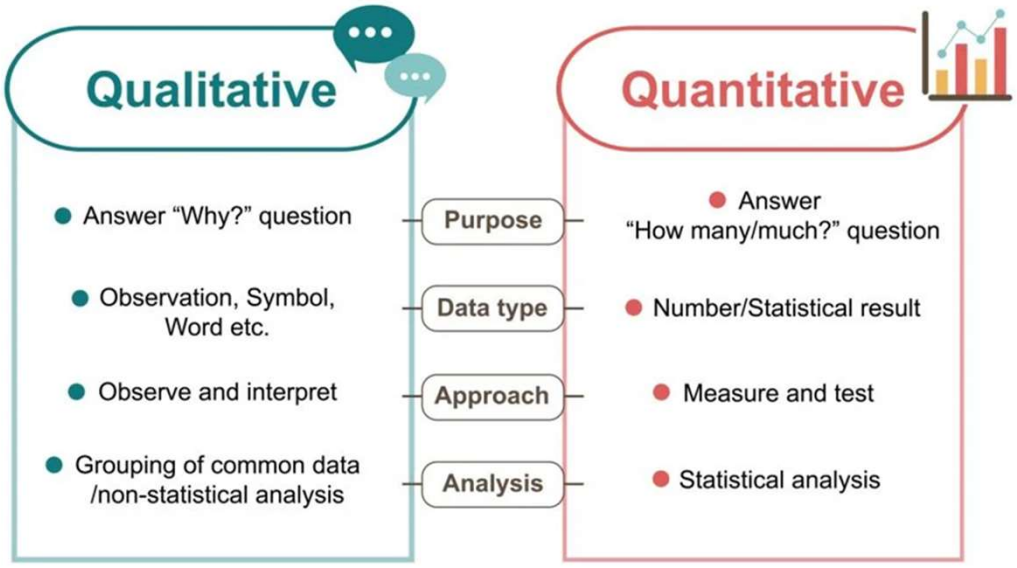


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

Data collection

"...a systematic process of gathering, measuring, and analyzing information from various sources to answer research questions, test hypotheses, or evaluate outcomes."

Why?

- **Insight and decision making:** Proper data collection leads to evidence-based conclusions that can be generalized or applied in applications. **(debugging)**
- **Testing hypotheses and theories:** Data allows you to test your hypotheses by providing measurable evidence.
    - Empirical evidence supports or challenges existing theories. **(...more debugging)**

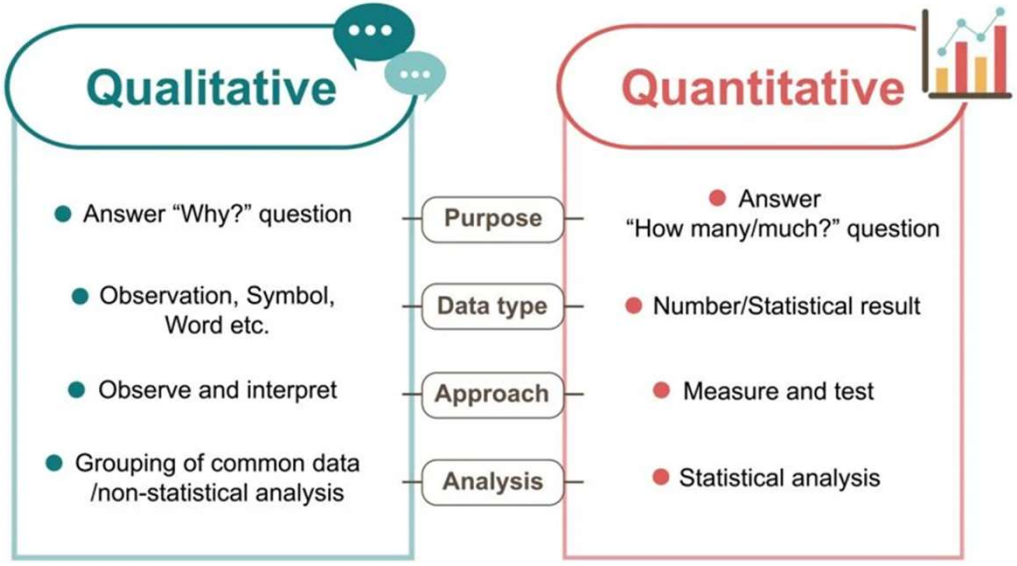


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

Data collection

"...a systematic process of gathering, measuring, and analyzing information from various sources to answer research questions, test hypotheses, or evaluate outcomes."

Why?

- **Replicability and transparency:** Well-documented and systematic data collection ensures transparency and helps others to replicate your work.

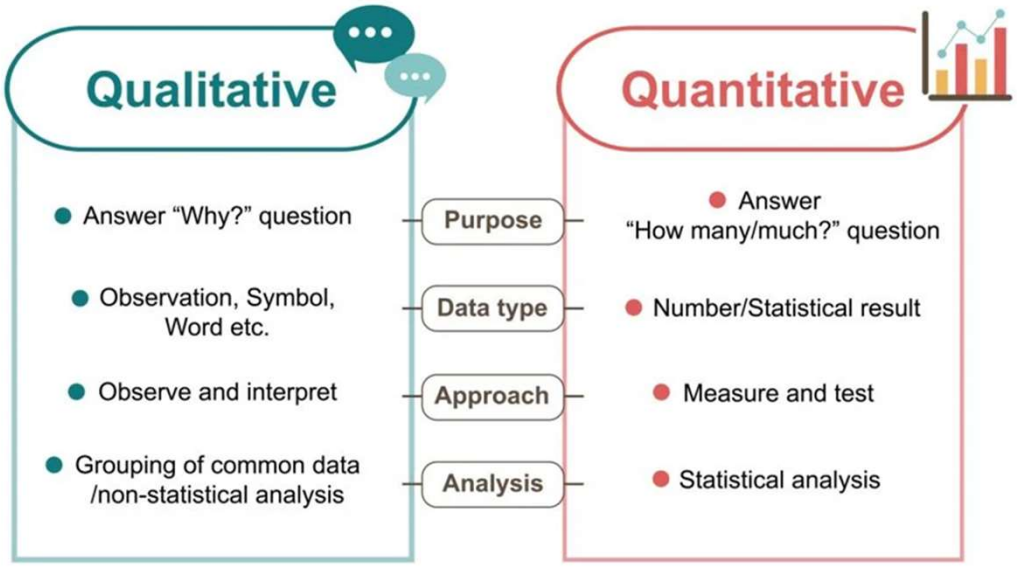


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

Data collection

"...a systematic process of gathering, measuring, and analyzing information from various sources to answer research questions, test hypotheses, or evaluate outcomes."

Why?

- **Replicability and transparency:** Well-documented and systematic data collection ensures transparency and helps others to replicate your work.
- **Documentation:** Proper documentation ensures that anyone who reads your results, understands what you have tested and why, which (of course) also involves data collection.

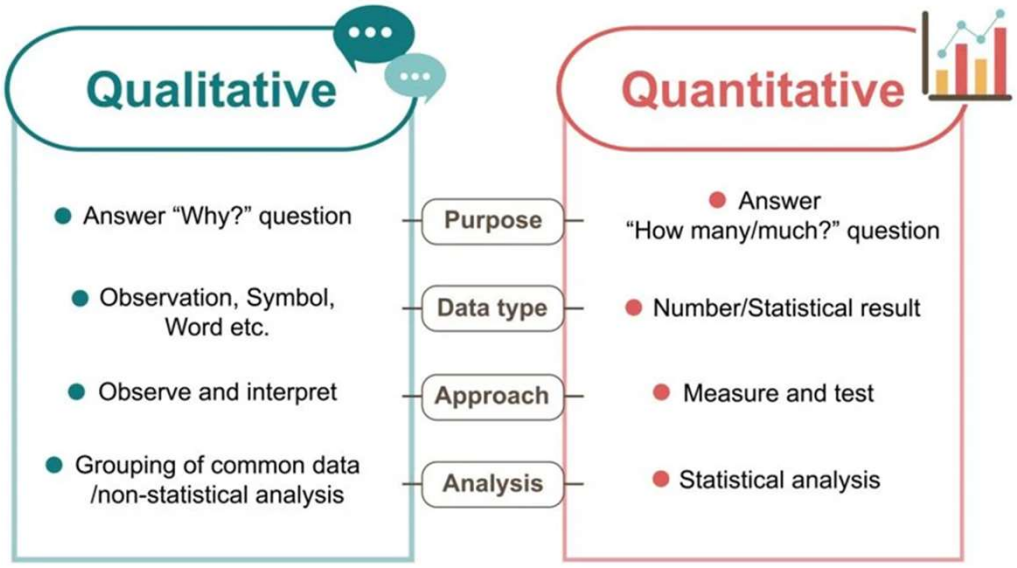


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

Data collection

"...a systematic process of gathering, measuring, and analyzing information from various sources to answer research questions, test hypotheses, or evaluate outcomes."

Why?

- **Replicability and transparency:** Well-documented and systematic data collection ensures transparency and helps others to replicate your work.
- **Documentation:** Proper documentation ensures that anyone who reads your results, understands what you have tested and why, which (of course) also involves data collection.

(...for all you future projects and reports…)

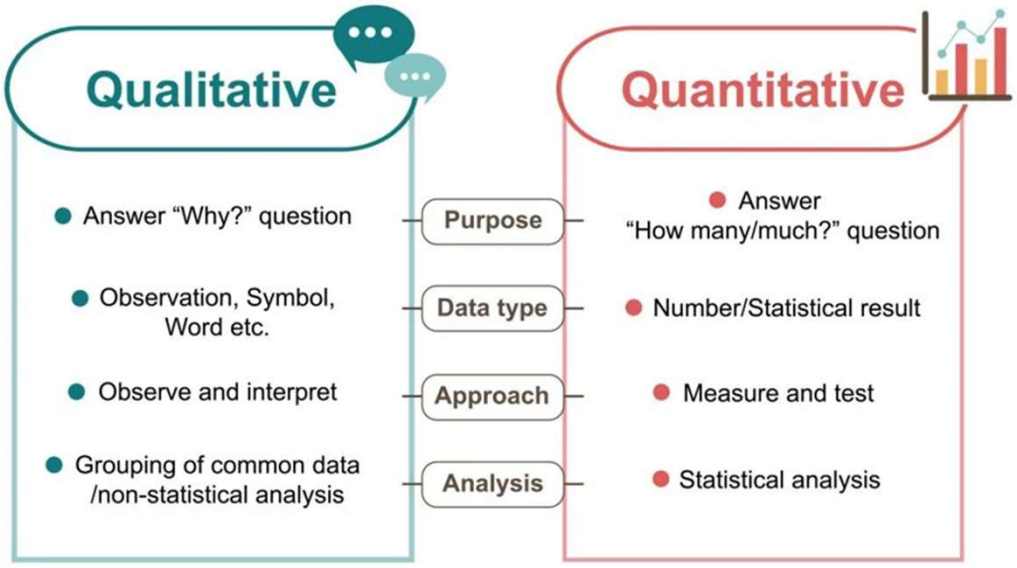
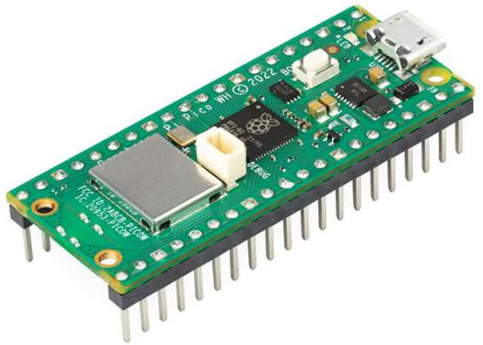


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

End-to-End Data Collection (using microcontrollers)

…a complete process, from

Sensor data acquisition → Data analysis



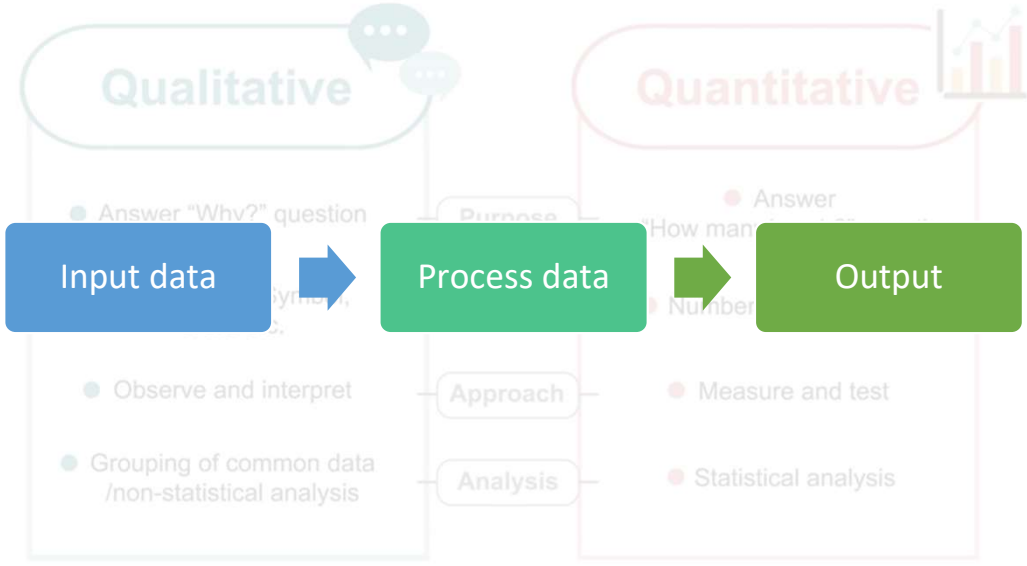
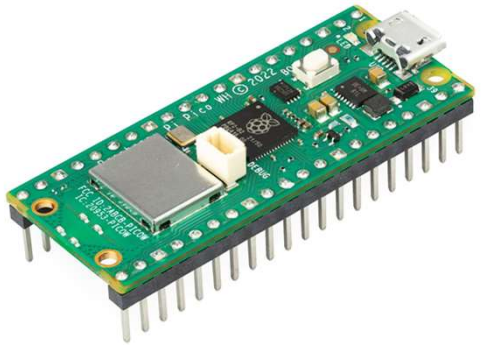


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

End-to-End Data Collection (using microcontrollers)



…a complete process, from

Sensor data acquisition → Data analysis

Roughly four steps:

1. **Define the objective**
2. **Design the data collection method**
3. **Testing and collect the data**
4. **Analyze and Interpret the Data**

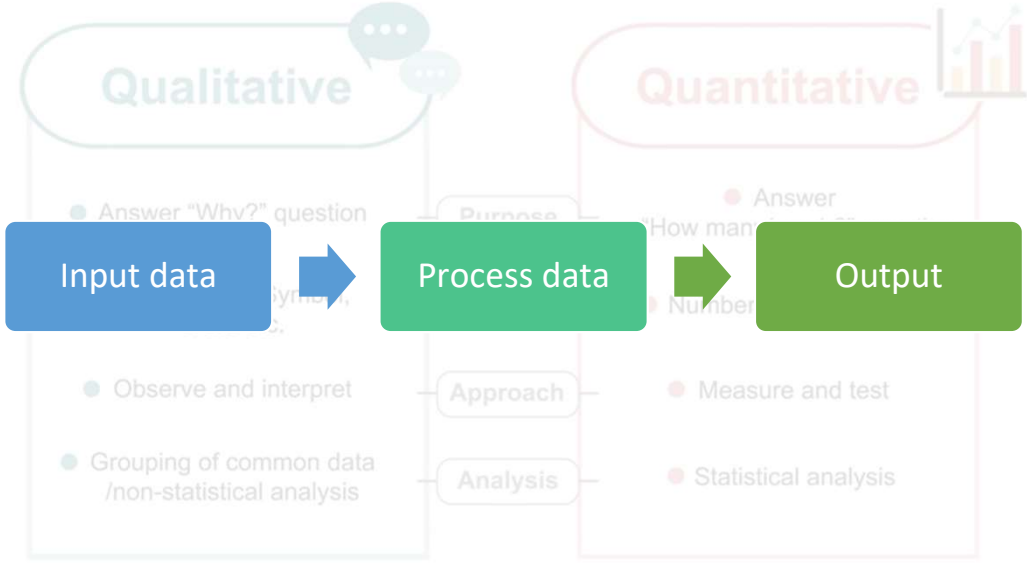
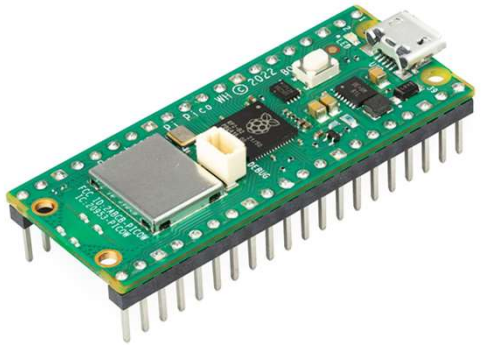


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

End-to-End Data Collection (using microcontrollers)



1. **Define the objective**
    a. Define the purpose of the data collection
        i. Why is this important?
        ii. Consider what you intend to discover or understand from the data?
        iii. What data do we need to gather to gain the in-sight that we wish?

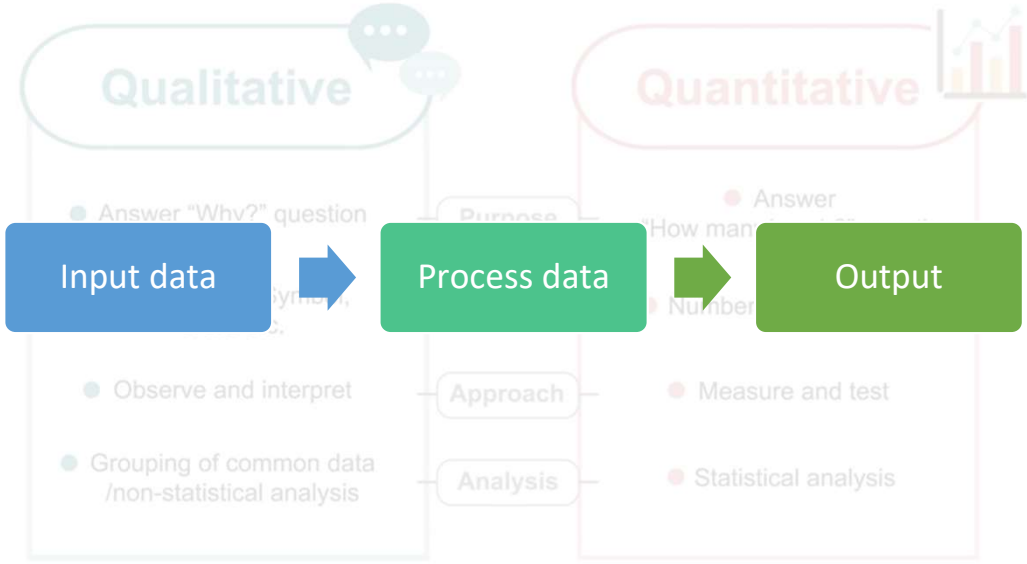


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

End-to-End Data Collection (using microcontrollers)

1. Define the objective

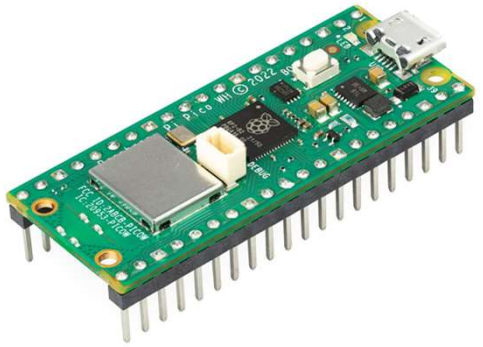
2. **Design the data collection method**
   a. Based on the objective, consider the design of how you will collect the data, including:
      i. **Experiment(s) / Setup:** controlled testing of hypotheses using the same setup and procedures

      ii. **Hardware**: Choice of sensor systems and devices

      iii. **Software:** Choice of software packages and interfaces, including:
         1. **Sample selection:** How many samples to ensure representativeness?

         2. **Numeric precision:**
            a. How many digits to ensure the precision?

            b. (Hardware) Can we even obtain the required precision?



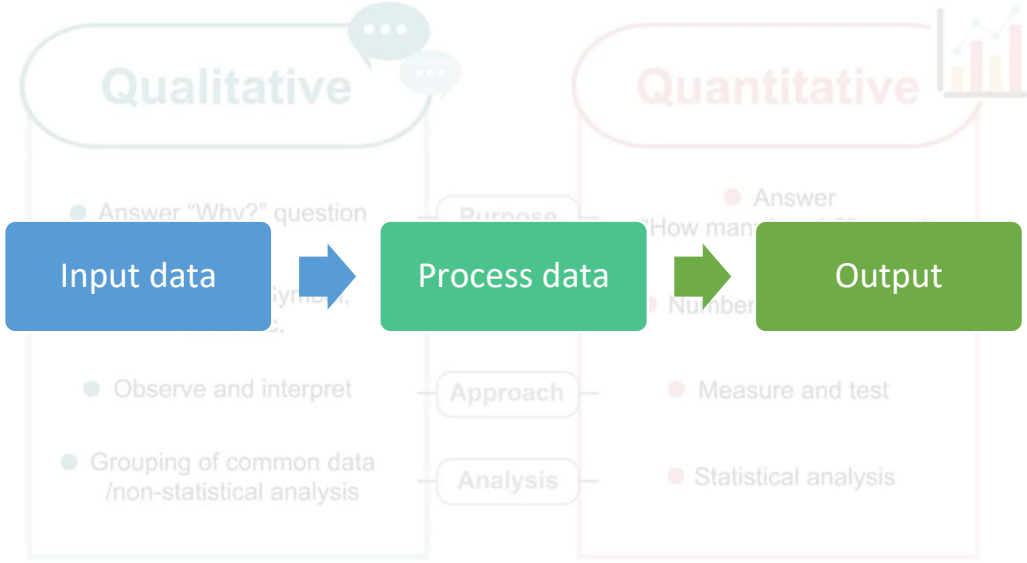


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

End-to-End Data Collection (using microcontrollers)

1. Define the objective

2. **Design the data collection method**
   a. Based on the objective, consider the design of how you will collect the data, including:
      i. **Experiment(s) / Setup:** controlled testing of hypotheses using the same setup and procedures

      ii. **Hardware**: Choice of sensor systems and devices

      iii. **Software:** Choice of software packages and interfaces, including:
         1. **Sample selection:** How many samples to ensure representativeness?

         2. **Numeric precision:**
            a. How many digits to ensure the precision?

   b. (Hardware) **Can we even obtain the required precision? As well as the Accuracy?**

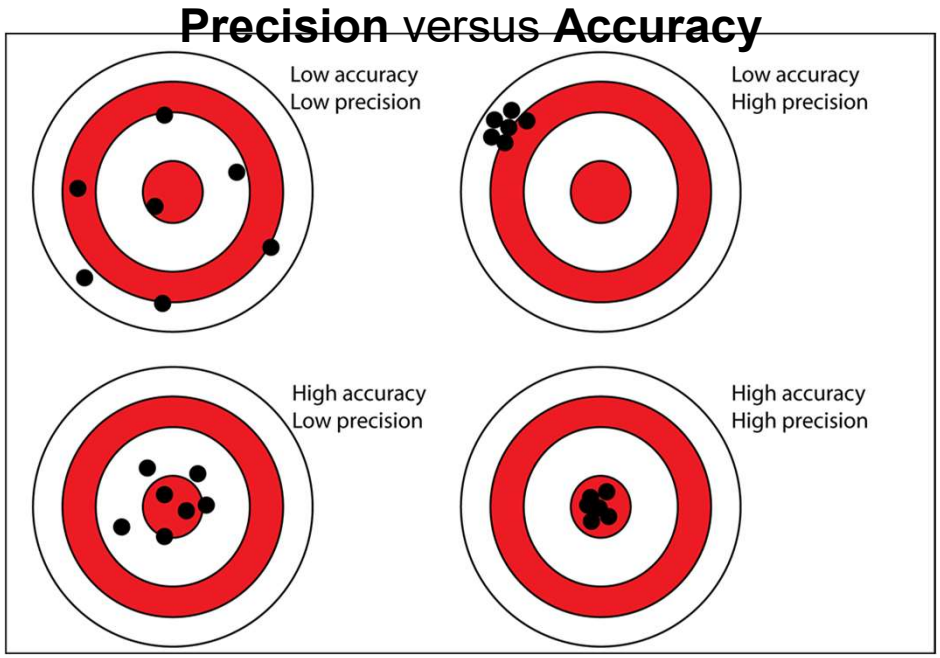


Illustration: https://www.antarcticglaciers.org/glacial-geology/dating-glacial-sediments-2/precision-and-accuracy-glacial-geology/

End-to-End Data Collection (using microcontrollers)

1. Define the objective

2. **Design the data collection method**
   a. Based on the objective, consider the design of how you will collect the data, including:
      i. **Experiment(s) / Setup:** controlled testing of hypotheses using the same setup and procedures

      ii. **Hardware**: Choice of sensor systems and devices

      iii. **Software:** Choice of software packages and interfaces, including:
         1. **Sample selection:** How many samples to ensure representativeness?

         2. **Numeric precision:**
            a. How many digits to ensure the precision?

            b. (Hardware) Can we even obtain the required precision? As well as the Accuracy?

Ground Truth data

"...accurate, verified data obtained through precise, often direct methods (such as calibrated instruments/sensors or reference standards) used to validate or calibrate the sensor readings."

Precision versus **Accuracy**

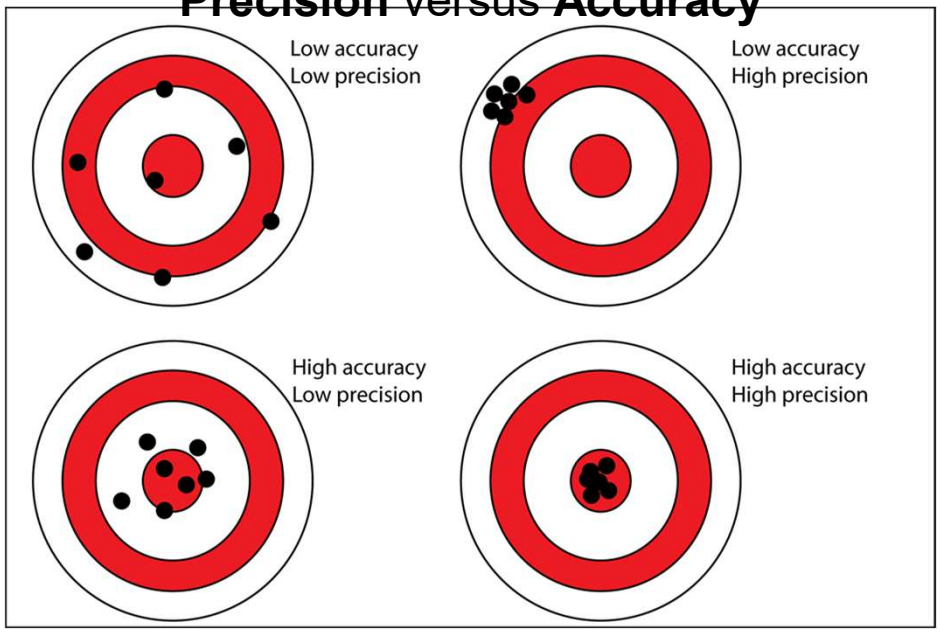


Illustration: https://www.antarcticglaciers.org/glacial-geology/dating-glacial-sediments-2/precision-and-accuracy-glacial-geology/

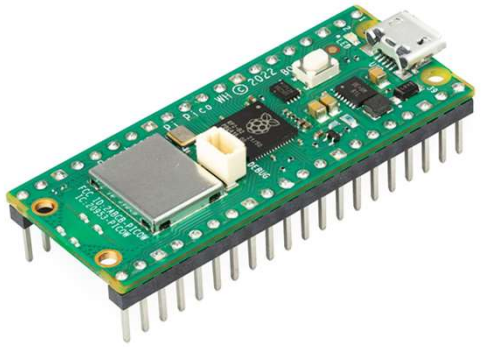
End-to-End Data Collection (using microcontrollers)

1. Define the objective

2. **Design the data collection method**
   a. Based on the objective, consider the design of how you will collect the data, including:
      i. **Software (continued):** Choice of software packages and interfaces, including:
         1. **Data Organization:** organize the data systematically, both:
            a. when collecting a single dataset (order of the data)
            b. when collecting multiple datasets in a row

         2. **Data storage:** Consider how the data should be stored:
            a. **Local Storage:** If data is stored locally, ensure that you don't get memory issues.

            b. **Wireless transmission:** Send data to a remote server, cloud storage, or another device.
               i. …which also means: **Security…**



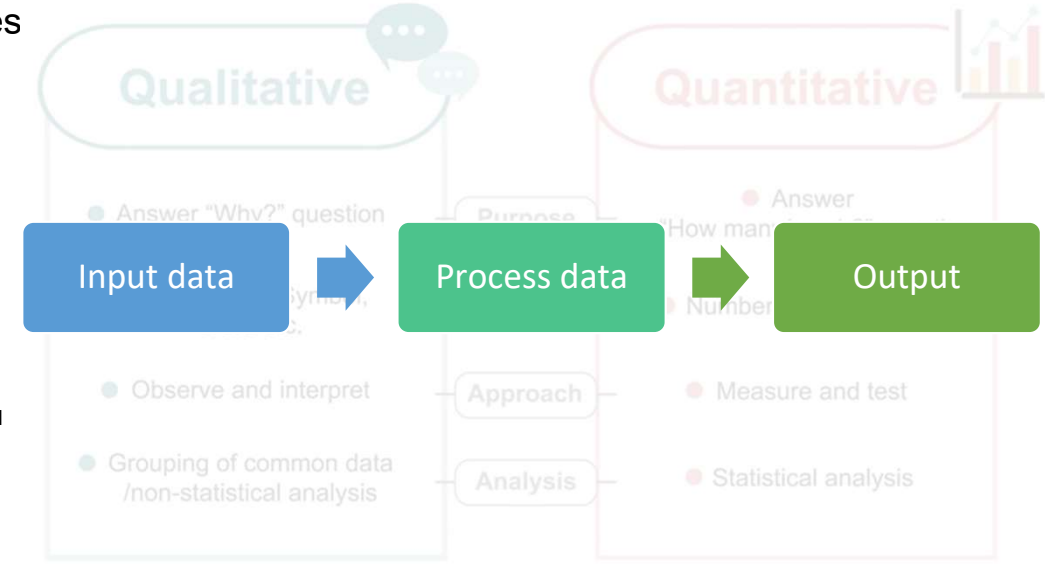


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

End-to-End Data Collection (using microcontrollers)

1. Define the objective

2. **Design the data collection method**
   a. Based on the objective, consider the design of how you will collect the data, including:
      i. **Software (continued):** Choice of software packages and interfaces, including:
         1. **Local Storage:** If data is stored locally, ensure that you don't get memory issues. **Either**
            a. Write to a local file (like the logging-example), or

            b. Write to a local file on another device (over serial)

Example: Reading internal temperature, and

```
import machine
import time

adcpin = 4
sensor = machine.ADC(adcpin)
def ReadTemperature():
   adc_value = sensor.read_u16()
   volt = (3.3/65535) * adc_value
   temperature = 27 - (volt - 0.706)/0.001721
   return round(temperature, 1)

print("time,temp")

while True:
   temperature = ReadTemperature()
   print("{},{}".format(time.time(),temperature))
   time.sleep(1)
```

Example: Storing the readings received over the serial

```
import serial
import time

# Open the serial port (adjust the port name and baud rate)
# Replace 'ttyACM0' with your serial port
ser = serial.Serial('/dev/ttyACM0', 9600)

# Open a file to write data
with open('data_log.csv', 'w') as f:
   while True:
      # Read a line from the serial port
      line = ser.readline().decode('utf-8').strip()
      print(line)  # Print to the terminal

      # Append the data to the file
      f.write(f'{line}\n')
      f.flush()  # Ensure data is written to the file
      time.sleep(1)
```

End-to-End Data Collection (using microcontrollers)

1. Define the objective

2. **Design the data collection method**
   a. Based on the objective, consider the design of how you will collect the data, including:
      i. **Software (continued):** Choice of software packages and interfaces, including:
         1. **Local Storage:** If data is stored locally, ensure that you don't get memory issues. **Either**
            a. Write to a local file (like the logging-example), or

            b. Write to a local file on another device (over serial)

CSV (Comma-Separated Values) versus **TXT (Plain Text File)**

- **CSV:** Suitable for collecting and storing structured data
- **TXT:** Suitable for storing unstructured or semi-structured data like logs

Example: Reading internal temperature, and

```
import machine
import time

adcpin = 4
sensor = machine.ADC(adcpin)
def ReadTemperature():
    adc_value = sensor.read_u16()
    volt = (3.3/65535) * adc_value
    temperature = 27 - (volt - 0.706)/0.001721
    return round(temperature, 1)

print("time,temp")

while True:
    temperature = ReadTemperature()
    print("{},{}".format(time.time(),temperature))
    time.sleep(1)
```

Example: Storing the readings received over the serial

```
import serial
import time

# Open the serial port (adjust the port name and baud rate)
# Replace 'ttyACM0' with your serial port
ser = serial.Serial('/dev/ttyACM0', 9600)

# Open a file to write data
with open('data_log.csv', 'w') as f:
    while True:
        # Read a line from the serial port
        line = ser.readline().decode('utf-8').strip()
        print(line)  # Print to the terminal

        # Append the data to the file
        f.write(f'{line}\n')
        f.flush()  # Ensure data is written to the file
        time.sleep(1)
```

End-to-End Data Collection (using microcontrollers)

1. Define the objective

2. **Design the data collection method**
   a. Based on the objective, consider the design of how you will collect the data, including:
      i. **Software (continued):** Choice of software packages and interfaces, including:
         1. **Local Storage:** If data is stored locally, ensure that you don't get memory issues. **Either**
            a. Write to a local file (like the logging-example), or

            b. Write to a local file on another device (over serial)

CSV (Comma-Separated Values) versus **TXT (Plain Text File)**

- **CSV:** Suitable for collecting and storing structured data
- **TXT:** Suitable for storing unstructured or semi-structured data like logs

JSON (JavaScript Object Notation))

Example: Reading internal temperature, and

```
import machine
import time

adcpin = 4
sensor = machine.ADC(adcpin)
def ReadTemperature():
    adc_value = sensor.read_u16()
    volt = (3.3/65535) * adc_value
    temperature = 27 - (volt - 0.706)/0.001721
    return round(temperature, 1)

print("time,temp")

while True:
    temperature = ReadTemperature()
    print("{},{}".format(time.time(),temperature))
    time.sleep(1)
```

Example: Storing the readings received over the serial

```
import serial
import time

# Open the serial port (adjust the port name and baud rate)
# Replace 'ttyACM0' with your serial port
ser = serial.Serial('/dev/ttyACM0', 9600)

# Open a file to write data
with open('data_log.csv', 'w') as f:
    while True:
        # Read a line from the serial port
        line = ser.readline().decode('utf-8').strip()
        print(line)  # Print to the terminal

        # Append the data to the file
        f.write(f'{line}\n')
        f.flush()  # Ensure data is written to the file
        time.sleep(1)
```

End-to-End Data Collection (using microcontrollers)

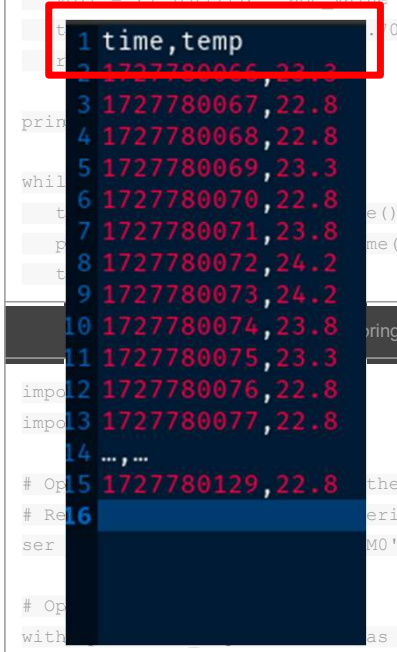
1. Define the objective

2. **Design the data collection method**

CSV (Comma-Separated Values)

- **CSV:** Suitable for collecting and storing structured data
  - **Header Row** (Optional)
    - The first line in a CSV file often contains column names, referred to as headers.
    - Each header describes the data field.
    - The header row is optional, but including it helps make the data easier to interpret.

Example: Reading internal temperature, and



time	temp
1727780066	23.3
1727780067	22.8
1727780068	22.8
1727780069	23.3
1727780070	22.8
1727780071	23.8
1727780072	24.2
1727780073	24.2
1727780074	23.8
1727780075	23.3
1727780076	22.8
1727780077	22.8
…	…
1727780129	22.8

End-to-End Data Collection (using microcontrollers)

1. Define the objective

2. **Design the data collection method**

CSV (Comma-Separated Values)

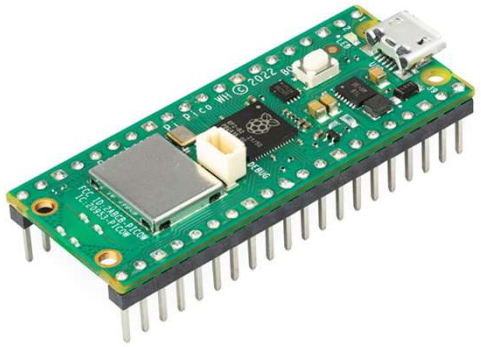
- **CSV:** Suitable for collecting and storing structured data
  - **Header Row** (Optional)
    - The first line in a CSV file often contains column names, referred to as headers.
    - Each header describes the data field.
    - The header row is optional, but including it helps make the data easier to interpret.
  - **Data Rows**
    - Each subsequent line represents a row of data.
    - Each value is separated by a comma (or other delimiter).
    - The position of each value corresponds to the columns defined in the header row.

Example: Reading internal temperature, and

```
time,temp
1727780066,23.3
1727780067,22.8
1727780068,22.8
1727780069,23.3
1727780070,22.8
1727780071,23.8
1727780072,24.2
1727780073,24.2
1727780074,23.8
1727780075,23.3
1727780076,22.8
1727780077,22.8
…,…
1727780129,22.8
```

time	temp
1727780066	23.3
1727780067	22.8
1727780068	22.8
1727780069	23.3
1727780070	22.8
1727780071	23.8
1727780072	24.2
1727780073	24.2
1727780074	23.8
1727780075	23.3
1727780076	22.8
1727780077	22.8
…	…
1727780129	22.8

End-to-End Data Collection (using microcontrollers)



1. Define the objective
2. Design the data collection method
3. **Testing and collect the data**
    a. **(if needed) Calibrate Sensors:** Ensure they provide accurate data.
    b. **(Initial) System testing:** Validate the setup
        i. Ensure it works under various conditions
            1. Reliability versus Robustness (over time)
        ii. Ensure that the correct data is collected (and in the correct format)
        iii. Ensure that the system stores it correct (if possible, even in case of sudden shutdown…)
    c. **(Actual) Data collection:** When tested and validated, begin collecting according to your plan

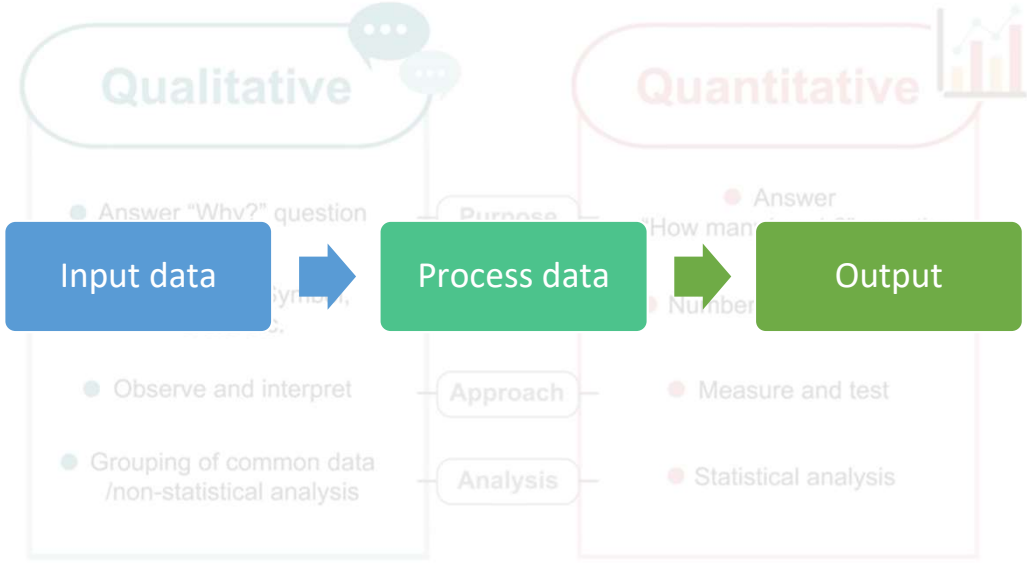
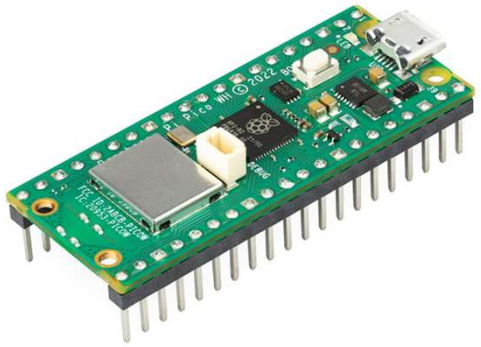


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

End-to-End Data Collection (using microcontrollers)



1. Define the objective
2. Design the data collection method
3. Testing and collect the data
4. **Analyze and Interpret the Data**
    a. **Data Retrieval:** First, gather the collected data
        i. **Local Storage:** from the external memory, eg. SD card, or the on-board memory.
        ii. **Wireless transmission:** Download/transfer from the cloud or device.
    b. **Data analysis:** Interpret the analyzed data in the context of the defined objectives.
        i. (*Partially* beyond scope) Statistics, such as
            1. Average, running average, standard deviation, etc.
        ii. **Data visualization**, for
            1. finding extremes, observe rate of change, etc.

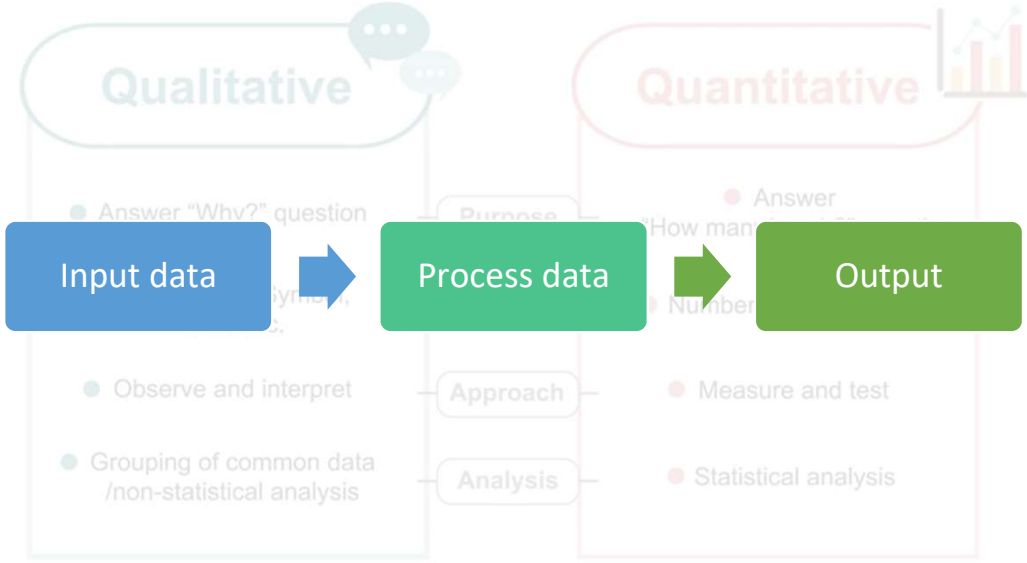


Illustration: https://www.simplypsychology.org/qualitative-quantitative.html

Visualization

Data visualization

“...a graphical representation of data and information.”

Why?

- **Better understanding:** Helps simplify complex datasets or large amounts of information by representing them in a visual format.
    - Makes a quick overview and understanding
    - *Humans* are (typically) better at processing visual information than raw data.

Data visualization

"...a graphical representation of data and information."

Why?

- **Better understanding:** Helps simplify complex datasets or large amounts of information by representing them in a visual format.
    - Makes a quick overview and understanding
    - *Humans* are (typically) better at processing visual information than raw data.
    - **Documentation:** Is essential for proper technical documentation of data or experiments.

"A picture is worth a thousand words"

*…but it always good to provide the raw data as well (**transparency**)…*

Data visualization

"...a graphical representation of data and information."

Why?

- **Decision-Making:** Decisions often rely on data or observations.
  - Visualized data enables more informed decisions **(data-driven decisions)**, since it will be based on the presented results.
    - …also for non-technical people to understand

- **Debugging and testing:** Can help identify anomalies or outliers that may indicate errors/bugs or areas requiring further investigation.
  - …especially when working with research and development

Data visualization

"...a graphical representation of data and information."

How? (most common data visualizations)

- **Charts and Graphs:**
  - Line plots,
  - Bar charts,
  - Histograms,
  - Scatter plots,
  - Bubble Charts,
  - Pie charts,
  - and many more…

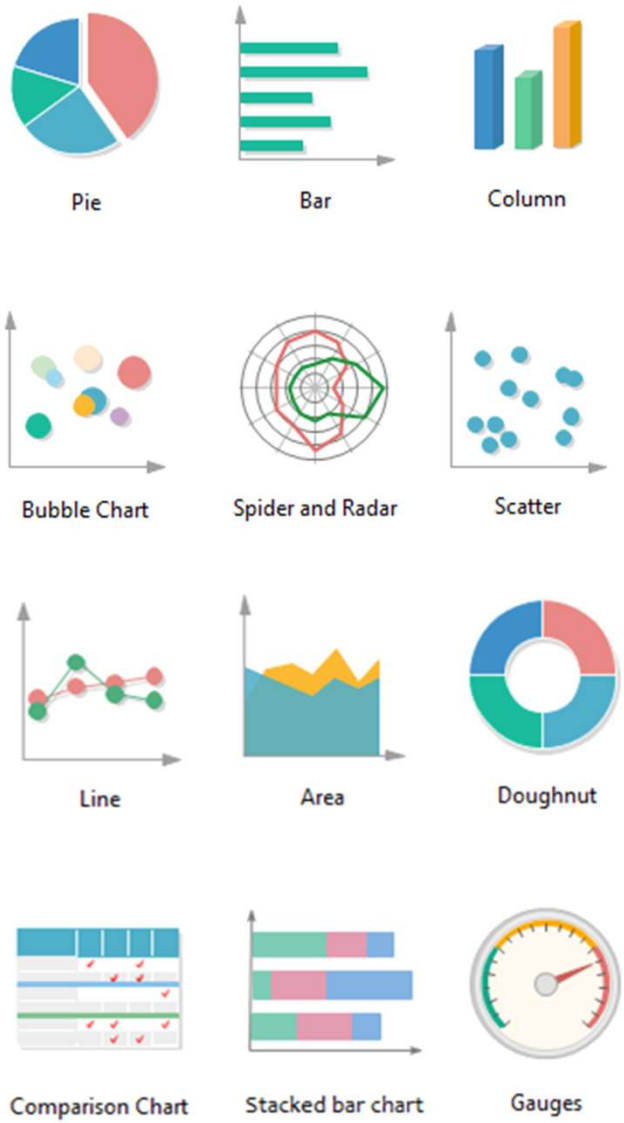


Illustration: https://medium.com/@Lynia_Li/as-you-know-there-are-many-types-of-charts-to-be-used-in-data-visualization-54da9b97092e

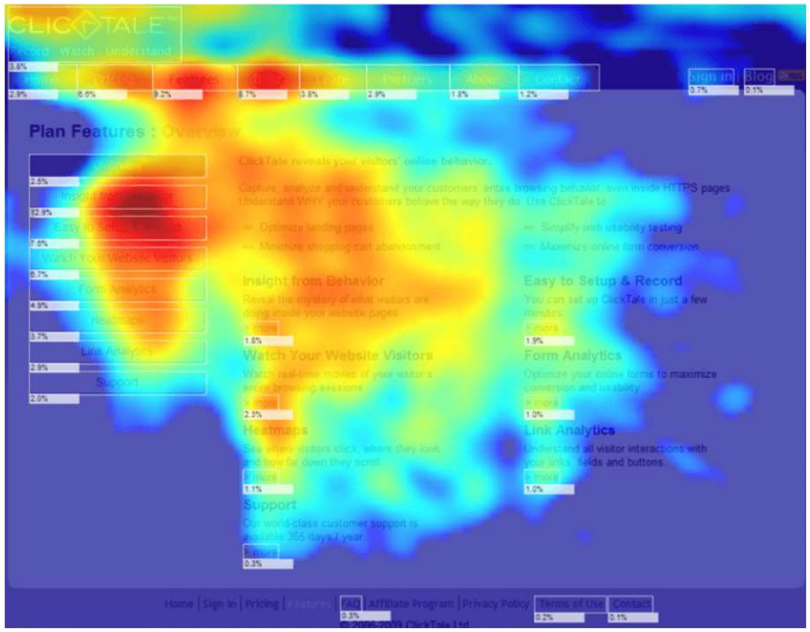
Data visualization

"...a graphical representation of data and information."

How? (most common data visualizations)

- **Charts and Graphs:**
  - Line plots,
  - Bar charts,
  - Histograms,
  - Scatter plots,
  - Bubble Charts,
  - Pie charts,
  - and many more…

- **Heatmaps:** Used to show the intensity of data in two dimensions.



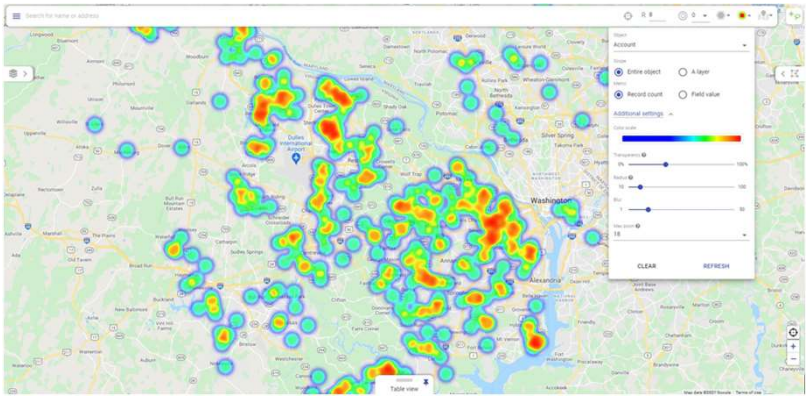


Illustration: https://jethrojeff.com/ and https://mapsly.com/heatmaps/

Data visualization

"...a graphical representation of data and information."

How? (most common data visualizations)

- **Charts and Graphs:**
  - Line plots,
  - Bar charts,
  - Histograms,
  - Scatter plots,
  - Bubble Charts,
  - Pie charts,
  - and many more…
- **Heatmaps:** Used to show the intensity of data in two dimensions.
- **Dashboards:** Interactive panels that display various visualizations, often used for real-time data monitoring.

Illustration: https://medium.com/@antonioneto_17307/is-a-dashboard-necessary-a3153db47219

Data visualization

"...a graphical representation of data and information."

How? (most common data visualizations)

- **Charts and Graphs:**
  - Line plots,
  - Bar charts,
  - Histograms,
  - Scatter plots,
  - Bubble Charts,
  - Pie charts,
  - and many more…
- **Heatmaps:** Used to show the intensity of data in two dimensions.
- **Dashboards:** Interactive panels that display various visualizations, often used for real-time data monitoring.
- **and many more… Which one to choose?**

Illustration: https://medium.com/@antonioneto_17307/is-a-dashboard-necessary-a3153db47219

Data visualization

"...a graphical representation of data and information."

How? (most common data visualizations)

- **Choosing the right data visualization**
  - Highly depends on several factors
    1. **The type of data**
       a. Is it continuous or discrete? Single or multi-dimensional? And does it involve relationships between variables?

    2. **What you want to convey**
       a. Do want to show a compare data sets? Or highlight an anomaly? Or demonstrate a correlation?
          i. How many details are needed?
          ii. Remember only to include the necessary data and highlight the important part to make it easier for the reader to understand it.

    3. **Who is the reader / audience:**
       a. **Background:** Technical knowledge? General knowledge?
       b. **Medie:** For a report? Or a presentation? For debugging?

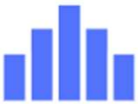
CHART TYPE	DATA TYPE
Line Chart 	Continuous or discrete
Bar Chart 	Discrete
Scatter Chart 	Continuous
Bubble Chart 	Relationship between three variables identifying trends
Histogram 	Distribution Analysis
Heatmap 	Density or distribution of data
Gantt Chart 	Project management, scheduling, task duration

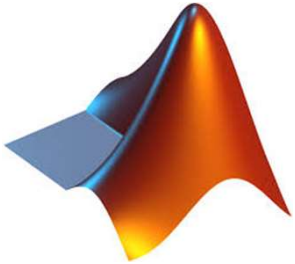
Illustration: https://medium.com/@ThisIsShubhangi/data-visualization-choosing-the-right-chart-matters-d4f6b75f8ae0

Data visualization

"...a graphical representation of data and information."

Tools?

- Excel / Spreadsheet
- Microsoft Power BI
- Google Sheets / Google Data Studio
- Wolfram Mathematica
- R (ggplot2)
- Matlab
- Dashboards / Cloud solutions
- Python libraries
- **…and many, many more…**



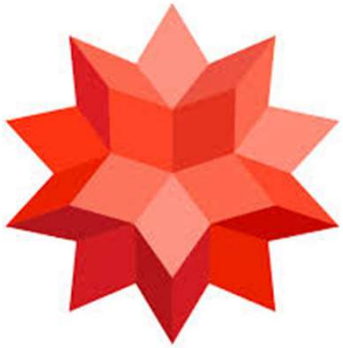


Illustration: from Google Images

Data visualization

"...a graphical representation of data and information."

Tools?

- Excel / Spreadsheet
- Microsoft Power BI
- Google Sheets / Google Data Studio
- Wolfram Mathematica
- R (ggplot2)
- Matlab
- Dashboards / Cloud solutions
- **…and many, many more…**

- **Data visualization** using Python ("one-stop-shop")
  - **Matplotlib** (https://matplotlib.org/)
  - **Plotly** (https://plotly.com/python/)
  - **Pandas** (http://pandas.pydata.org/docs/index.html)
  - **Seaborn** (https://seaborn.pydata.org/)
  - …and many, many more



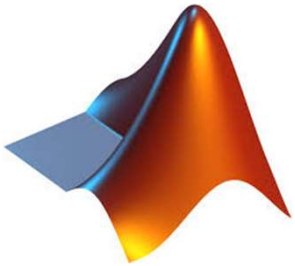
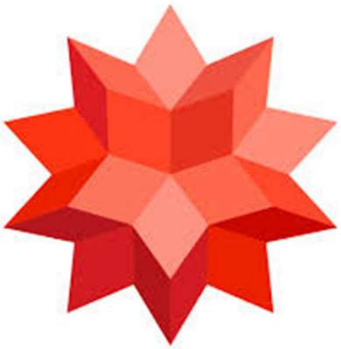




Illustration: from Google Images

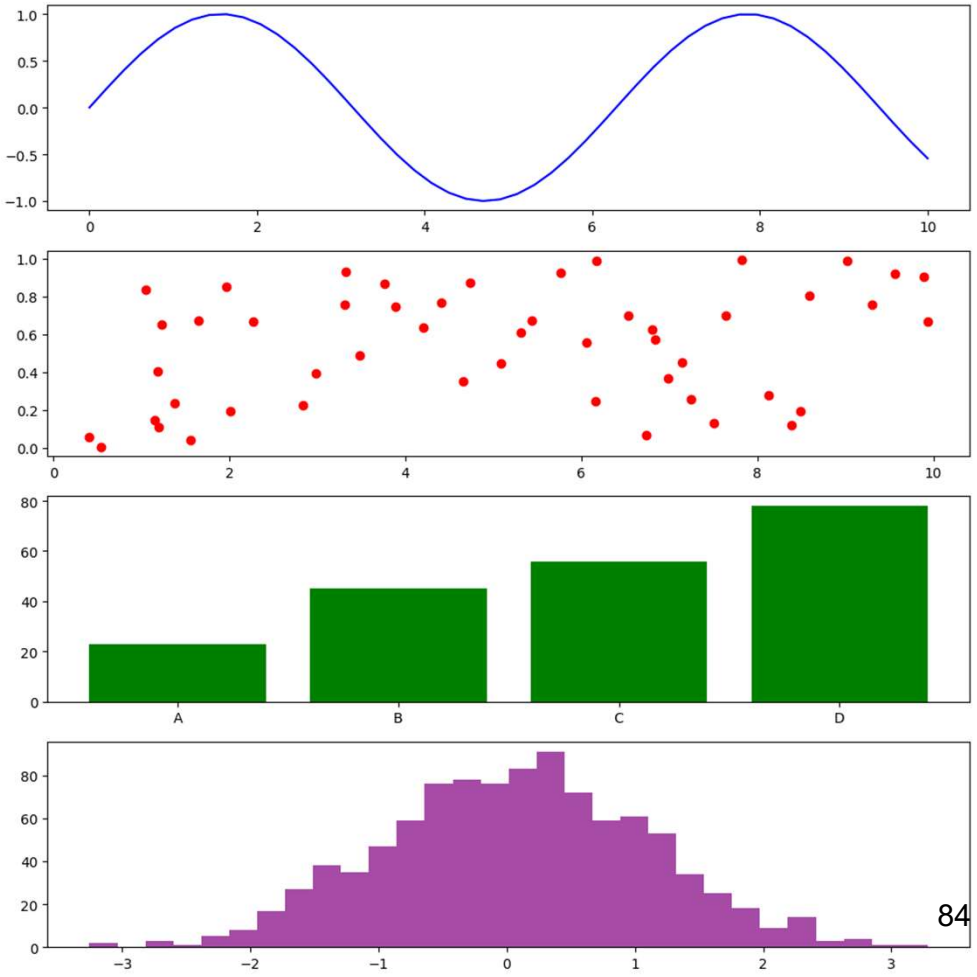
Data visualization

"...a graphical representation of data and information."

Data visualization using Python (**Matplotlib**)

- Source: https://matplotlib.org/stable/
- **Importing Matplotlib**
  - `import matplotlib.pyplot as plt`: This is the standard way to import the `pyplot` module from Matplotlib, which provides the functions needed for creating plots.

- **Basic Plots** (few examples)
  - **Line Plot (`plt.plot(x, y)`)**: Draws a line connecting data points `(x, y)`. Useful for visualizing trends over time or continuous data.
  - **Scatter Plot (`plt.scatter(x, y)`)**: Displays individual data points without connecting lines. Useful for visualizing the relationship between two variables.
  - **Bar Chart (`plt.bar(categories, values)`)**: Shows the comparison between different categories with rectangular bars. Useful for categorical data.
  - **Histogram (`plt.hist(data, bins)`)**: Creates a histogram with the data divided into the specified number of `bins`. Useful for visualizing the distribution of a dataset.



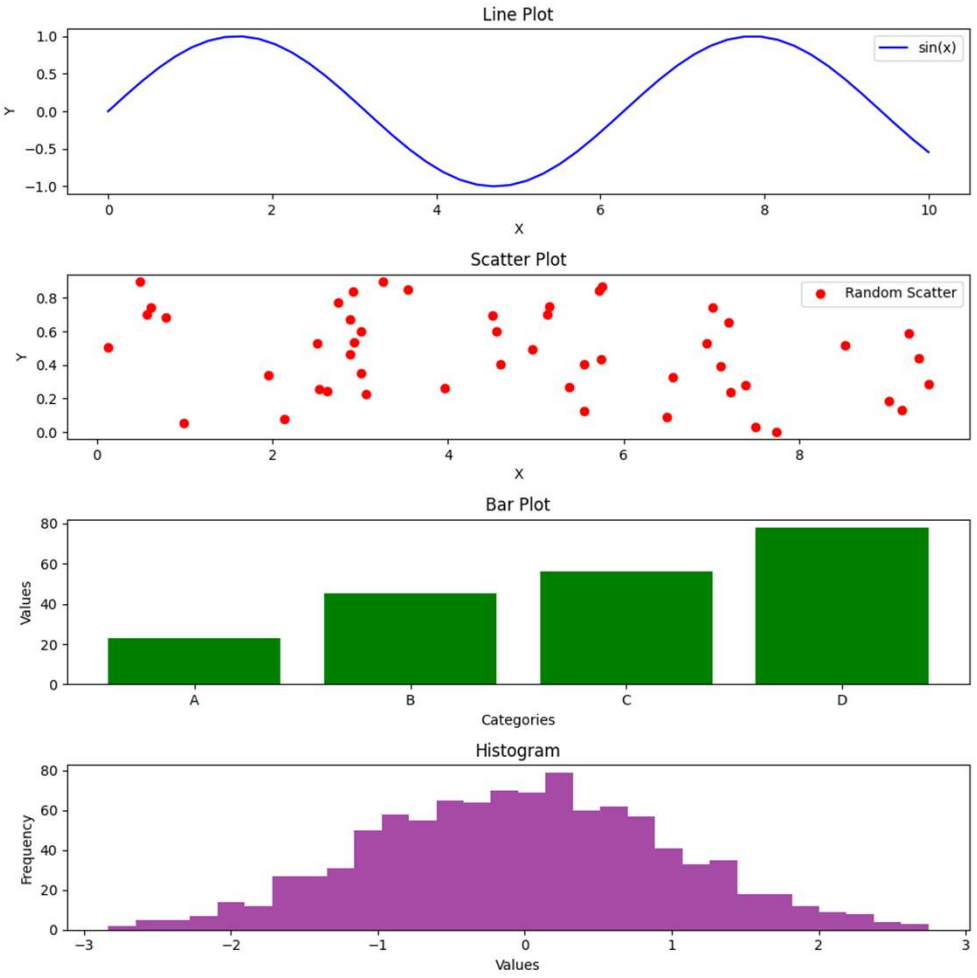
Data visualization

"...a graphical representation of data and information."

Data visualization using Python (**Matplotlib**)

- Source: https://matplotlib.org/stable/

- **Customizing Plots**
  - Titles and Labels (`plt.title()`, `plt.xlabel()`, `plt.ylabel()`): Adds context to the plot by setting a title and labels for the x-axis and y-axis.
  - Line Styles and Colors (`plt.plot(x, y, color='green', linestyle='--', marker='o')`): Customizes the appearance of the plot, including line color, style, and markers.
  - Legend (`plt.legend()`): Displays a legend on the plot to identify multiple data series.



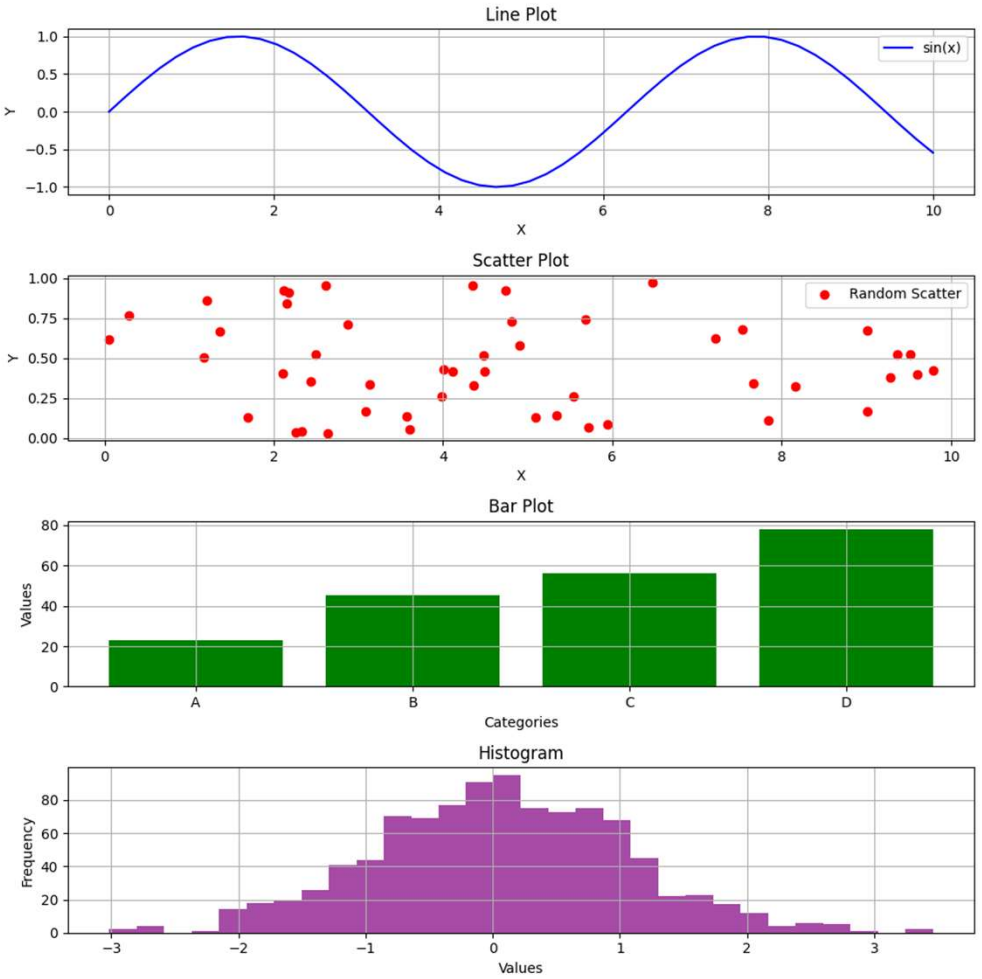
Data visualization

"...a graphical representation of data and information."

Data visualization using Python (**Matplotlib**)

- Source: https://matplotlib.org/stable/

- **Customizing Plots**
  - Grid Lines (`plt.grid(True)`): Adds grid lines to the plot, making it easier to read and interpret data points.
  - Adjusting Axis Limits (**`plt.xlim(min, max)`** / **`plt.ylim(min, max)`**): Manually sets the limits of the x-axis or y-axis, controlling the range of data that is displayed.
  - Plot size (`figsize=(x, y)`: Sets the figure size to `x` inches wide and `y` inches tall.
    - Ensure that you can control the dimensions of your plots to make them more readable
    - …and useful when preparing your visualizations for a report or presentation.



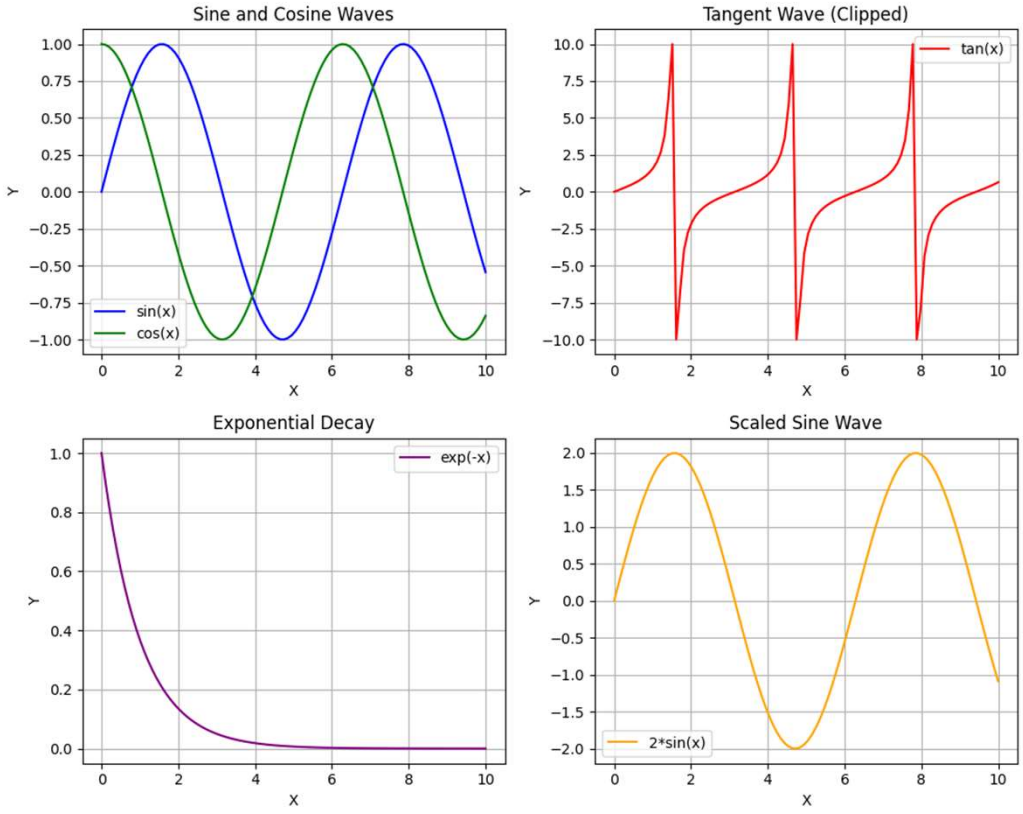
Data visualization

"...a graphical representation of data and information."

Data visualization using Python (**Matplotlib**)

- Source: https://matplotlib.org/stable/

- **Multiple plots**
  - Multiple plots on the same figure: You can plot multiple lines or datasets on the same figure by simply calling the `plot()` function multiple times before using `plt.show()`.
    - Remember to use legends (`plt.legend()`): Displays a legend on the plot to identify multiple data series.
    - …and must (typically) be of the same type
  - Subplots (grid layout): You can create a grid of subplots in a single figure using `plt.subplot()` or `plt.subplots()`.
    - Can be different types
  - (...and of course combining subplots of multiple plots in the same figure…)



SDU

Data visualization

“...a graphical representation of data and information.”

Automation

- **Saving the figures (**`plt.savefig('file_name.png')`): Saves the current figure to a file named `file_name.png` in the current working directory.

- Parsing data
  - Hard-coded values
    - …but what if it is > 2,000 data points?

  - Read from a Comma-Separated Values (CSV) (or Text File Format (TXT)) file
    - Step-by-Step
      1. **Read the CSV File**: Use the `pandas` library to read the CSV file.

      2. **Extract Data**: Extract the `time` and `temperature` columns from the DataFrame.

      3. **Plot the Data**: Use the matplotlib to plot the temperature over time

      4. **Save the Plot:** Use the `savefig` for automatically saving the figure locally.

Data visualization

"...a graphical representation of data and information."

Example: Collecting data from the internal temperature sensor

1. **Define the objective:**

   Does the frequency affect the internal temperature?

1. **Design the data collection method**
   a. **H/W:** Raspberry Pi Pico | **S/W:** → (example on the right)
   b. **Samples:** 60 (once a second)
   c. **Storage:** locally as a .csv-file

2. **Testing and collect the data**
   a. ***[33, 100, 150, 200, 250]*** MHz

3. **Analyze and Interpret the Data**

Example: Use Wi-Fi to get the current time

```
import machine
import time

adcpin = 4
sensor = machine.ADC(adcpin)

freq = 33
machine.freq(freq*1_000_000)

# Get the current time
current_time = time.localtime()
# Print the date and time in a readable format
log_time = "Date and time: {:04d}-{:02d}-{:02d} {:02d}:{:02d}:{:02d}".format(
    current_time[0], current_time[1], current_time[2],  # Year, Month, Day
    current_time[3], current_time[4], current_time[5]   # Hour, Minute, Second
)

def ReadTemperature():
    adc_value = sensor.read_u16()
    volt = (3.3/65535) * adc_value
    temperature = 27 - (volt - 0.706)/0.001721
    return round(temperature, 1)

# Open a file in append mode
with open("data/temp_log_{}_mhz.csv".format(freq), "w") as log_file:
    log_file.write("cnt, temp\n")
    for cnt in range(60):
        temperature = ReadTemperature()
        log_entry = "{},{}\n".format(cnt,temperature)
        print(log_entry)  # Print the temperature to the console

        # Write the log entry to the file
        log_file.write(log_entry)
        log_file.flush()  # Ensure the data is written to the file

        time.sleep(1)

    log_file.close()
```

Data visualization

"...a graphical representation of data and information."

Example: Collecting data from the internal temperature sensor

1. **Define the objective:**

   Does the frequency affect the internal temperature?

1. **Design the data collection method**
   a. **H/W:** Raspberry Pi Pico | **S/W:** → (example on the right)
   b. **Samples:** 60 (once a second)
   c. **Storage:** locally as a .csv-file

2. **Testing and collect the data**
   a. ***[33, 100, 150, 200, 250]*** MHz

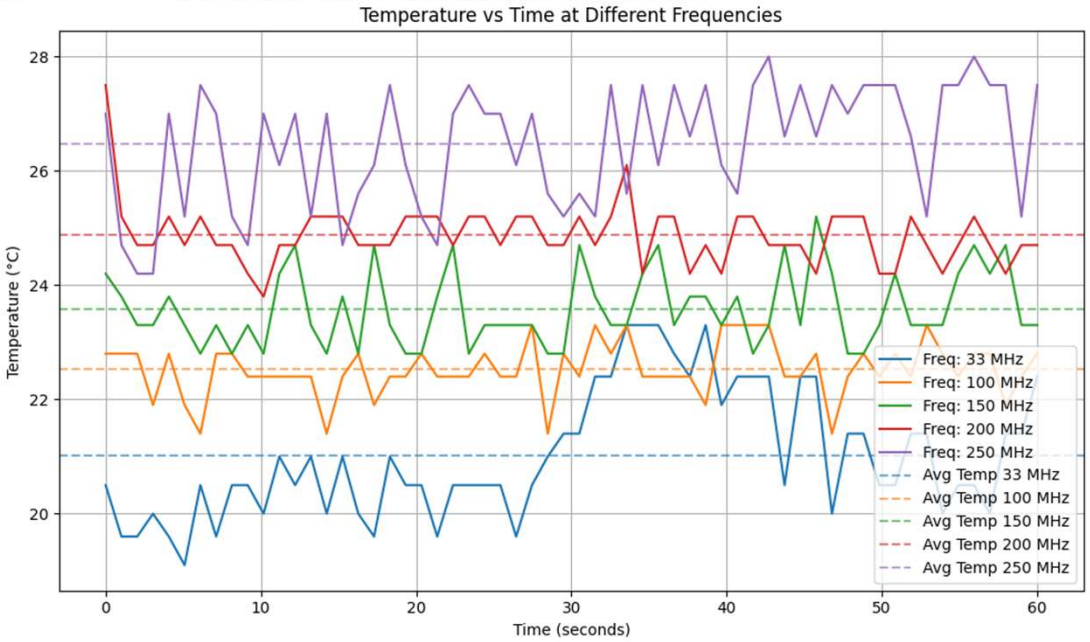
3. **Analyze and Interpret the Data**

33 MHz		100 MHz		150 MHz		200 MHz		250 MHz	
0	**20.5**	0	**22.8**	0	**24.2**	0	**27.5**	0	**27.0**
1	**19.6**	1	**22.8**	1	**23.8**	1	**25.2**	1	**24.7**
2	**19.6**	2	**22.8**	2	**23.3**	2	**24.7**	2	**24.2**
3	**20.0**	3	**21.9**	3	**23.3**	3	**24.7**	3	**24.2**
4	**19.6**	4	**22.8**	4	**23.8**	4	**25.2**	4	**27.0**
5	**19.1**	5	**21.9**	5	**23.3**	5	**24.7**	5	**25.2**
6	**20.5**	6	**21.4**	6	**22.8**	6	**25.2**	6	**27.5**
7	**19.6**	7	**22.8**	7	**23.3**	7	**24.7**	7	**27.0**
8	**20.5**	8	**22.8**	8	**22.8**	8	**24.7**	8	**25.2**
9	**20.5**	9	**22.4**	9	**23.3**	9	**24.2**	9	**24.7**
10	**20.0**	10	**22.4**	10	**22.8**	10	**23.8**	10	**27.0**
11	**21.0**	11	**22.4**	11	**24.2**	11	**24.7**	11	**26.1**
12	**20.5**	12	**22.4**	12	**24.7**	12	**24.7**	12	**27.0**
13	**21.0**	13	**22.4**	13	**23.3**	13	**25.2**	13	**25.2**
14	**20.0**	14	**21.4**	14	**22.8**	14	**25.2**	14	**27.0**
15	**21.0**	15	**22.4**	15	**23.8**	15	**25.2**	15	**24.7**
16	**20.0**	16	**22.8**	16	**22.8**	16	**24.7**	16	**25.6**
17	**19.6**	17	**21.9**	17	**24.7**	17	**24.7**	17	**26.1**
18	**21.0**	18	**22.4**	18	**23.3**	18	**24.7**	18	**27.5**
19	**20.5**	19	**22.4**	19	**22.8**	19	**25.2**	19	**26.1**
20	**20.5**	20	**22.8**	20	**22.8**	20	**25.2**	20	**25.2**
…	**…**	…	**…**	…	**…**	…	**…**	…	**…**
58	**21.4**	58	**22.4**	58	**23.3**	58	**24.7**	58	**25.2**
59	**22.4**	59	**22.8**	59	**23.3**	59	**24.7**	59	**27.5**

Data visualization

"...a graphical representation of data and information."

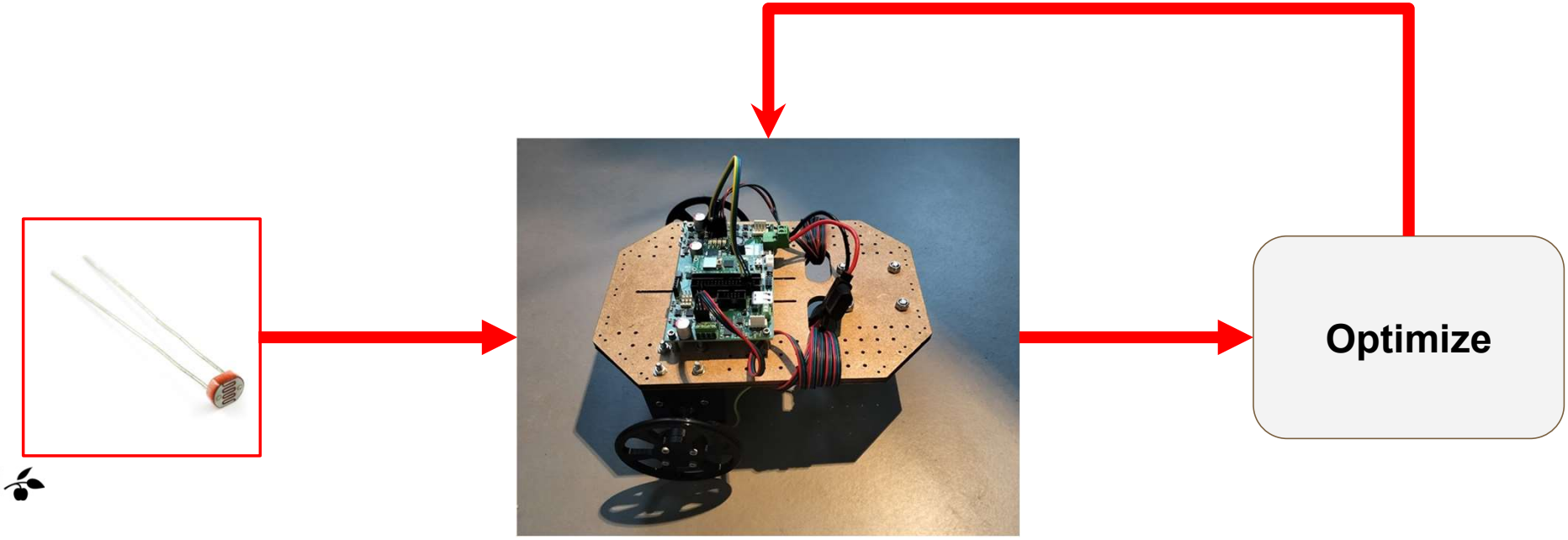
Example: Collecting data from the internal temperature sensor



Example: Use Wi-Fi to get the current time

```
import matplotlib.pyplot as plt
import numpy as np

# Corresponding to machine.freq() values
frequencies = [33, 100, 150, 200, 250]

# Time matching the length of the temperature datasets, 60 points)
time = np.linspace(0, 60, 60)

# Temperature data
temperature_data = [
    [20.5,19.6,19.6,20.0,19.6, ... ,22.4],
    [22.8,22.8,22.8,21.9,22.8, ... ,22.8],
    [24.2,23.8,23.3,23.3,23.8, ... ,23.3],
    [27.5,25.2,24.7,24.7,25.2, ... ,24.7],
    [27.0,24.7,24.2,24.2,27.0, ... ,27.5]
]

# Create a figure
plt.figure(figsize=(10, 6))

# 2.1: Plot temperature data for each frequency
plt.plot(time, temperature_data[0], label=f'Freq: {frequencies[0]} MHz') # Plot for 33
MHz

# 2.4: Calculate and plot the average temperature for each frequency
# - Ensure that the color matches the

# 2.2: Add appropriate plot titles, labels, legends, and grid for better visualization.
# - Add a appropriate title
# - Add labels
# - Add a legend
# - Add a grid

# Show the plot
plt.tight_layout()
plt.show()

# 2.3: Save the plot as an image file
```

Portfolio 3: Integrating a Light Dependent Resistor (LDR) on a Mobile Robot



Assignments