General

MicroPython for Beginners

MicroPython for Beginners info

About the course

MicroPython Primer is a Free Course is Offered by NIELIT Calicut and Supported by SMART LAB Project under Chips to Start-Up (C2S) Program of MeitY, Govt of India.

Course Duration and Schedule:

- **Start Date: 25th June 2025**
- **End Date: 29th June 2025**
- **Total Duration: 5 hours (1 hour per day)**
- **Mode of lectures: Pre-recorded**
- **Hardware Lab Access: 1hr**
- **Lecture Upload Time: 10 AM daily**
- **Flexible Viewing: Lectures can be watched anytime during the course schedule**

Eligibility:

- **Open to anyone interested to learn Embedded Systems**

Lab Access:

- **Lab Access Date: Will be announce later**

Participation certificate:

Those who are satisfying any one of the following conditions is eligible for participation certificate within two weeks after completion of the course.

1. **Attendance: 60% through the online portal (Login for the day will be considered as attendance)**

   Or

2. **Exit test with 50% marks**

Remote lab access is not a mandatory criteria for participation certificate.

Exit Test Details:

1. **Format: Multiple Choice Questions (MCQs)**
2. **Duration: 15 minutes**
3. **Number of Questions: 10, each carrying 1 mark**
4. **Passing Marks: Minimum 5 marks**
5. **Attempts Allowed: Only one attempt**
6. **Recommendation: Ensure good internet connectivity during the test**

Exit Test Availability:

- **From: 30th June 2025**
- **To: 1st July 2025**

Important Dates:

Course Starting Date	**25-06-2025**
Course Ending Date	**29-05-2025**

Certified Candidates List Announcement Date	**02-07-2025**
Tentative Certificate Distribution Date	**10-07-2025**
Lectures Availability	**20-07-2025**

FAQ

✔ Done

Smart Lab free courses - FAQ

Schedule

✔ Done

Learning path

✔ Done

Discussion Forum

✔ Done

Post your doubts here.

Day1-Introduction to course and basics of MicroPython

D1-S1-Course introduction

✔ Done

D1-S2-Overview of MicoPython

✔ Done

Python IDE

✔ Done

Day2-Data types- Loops-conditional statements

D2-S1-Variables and data types

✔ Done

D2-S2Getting started with MicroPython

✔ Done

Python lab exercises-I

✔ Done

Python lab exercises-I

Python lab exercises-II

✔ Done

Python lab exercises-II

Day3-Functions / classes and objects - Modules

D3-S1-Python-Functions

✔ Done

D3-S2 Python Class and Objects.

✔ Done

D3-S3-Python-Modules

✔ Done

Example_codes

Day4-Overview of microcontrollers and Embedded systems

D4-S1-Overview of Microcontroller and Embedded Systems

D4-S2-Introduction to RPi-W

Datasheets

Datasheets

- Those who are interested in using Remote Hardware Lab can mark this Activity as Done
- Those who are opted lab can only book convenient lab slot timing. (Will be informed later.)
- Remote hardware lab usage is not considered for getting certificate and attendance.
- Remote hardware lab schedule. will be informed through mail.

✔ Done

Day5-Developing application with Micropython on RPi PICO-W

D5-S1-RPi Pico -W -Working with GPIO

D5-S2-Application development-PWM

Resources

Remote Hadware Lab Access - Demonstration

How to use remote lab

✔ Done

Exit test

Opened: Monday, 30 June 2025, 12:00 AM
Closes: Tuesday, 1 July 2025, 11:59 PM

✔ Done

Course Feedback

Opened: Monday, 30 June 2025, 12:00 AM
Closes: Tuesday, 1 July 2025, 11:59 PM

रा.इ.सू.प्रौ.सं NIELIT | राष्ट्रीय इलेक्ट्रॉनिकी एवं सूचना प्रौद्योगिकी संस्थान
National Institute of Electronics & Information Technology

Ministry of Electronics & Information Technology
Government of India

Overview of Micropython & Applications

RAJESH M.

Agenda

- What is MicroPython?
- Python Vs MicroPython
- Features of MicroPython
- MicroPython Hardware Support
- MicroPython Applications
- CircuitPython vs MicroPython

-18:47 1x

What is MicroPython?

- Basically, Micropython is a lean and efficient implementation of the Python programming language.
- It is tiny and open source that runs on small embedded development boards which are your microcontrollers!
  - Developed by Damien George.
  - Designed for resource-constrained devices.
  - Open-source and community-driven.
- Micropython allows you to write clean and simple Python code to control your boards instead of using other more complicated languages like C++.

Python Vs MicroPython

- Micropython has a smaller standard library
- Micropython only has a small subset of the Python standard library.
- Micropython are designed to work under constrained conditions.
- Micropython allows modules to access low-level hardware. (eg. libraries to easily access and interact with GPIOs).
-

Key features

- Small memory footprint.
- Supports interactive REPL (Read-Eval-Print Loop).
- Supports Python 3 syntax.
- Extensible with C/C++ modules.
- Support inline assembler
- Compilation on the chip
- Cross-platform (Windows, macOS, Linux).

Hardware Support

- Raspberry Pi Pico
- ESP8266 and ESP32
- PyBoard
- Adafruit CircuitPython boards
- Many more microcontrollers and development boards.

Benefits of Scripting Language

- Learnability
- Rapid Prototyping
- Time to market
- Easy extensibility by a user
- Natural sandbox
- Security of extendability by a user
- Extension code, to maintain product integrity and protection against attack vectors,

Target Application areas

- IoT (Internet of Things) projects
- Home automation
- Robotics
- Wearable devices
- Sensor data acquisition
- Rapid prototyping

MicroPython for Product Development

Pro	Con
Productivity	Increased Hardware resources
Traceability	Lack of developer skills regarding scripting languages
Portability	
Licensing	
Support	

What Micropython Can't do

- Really small MCU's use traditional 'C'
- Dynamically typed language
- Memory fragmentation
- Embedded Linux System for Large projects

CircuitPython Vs MicroPython

CircuitPython:

- CircuitPython is designed with a strong focus on ease of use and beginner-friendliness.
- It is developed by Adafruit and aims to make programming microcontrollers as simple as possible.
- The emphasis is on accessibility and education.
- DIY applications

MicroPython:

- MicroPython, while also beginner-friendly, places a bit more emphasis on performance and efficiency.
- It aims to provide a more complete Python environment and has a broader user base, including both beginners and experienced developers.

Program Development Flow

High level language

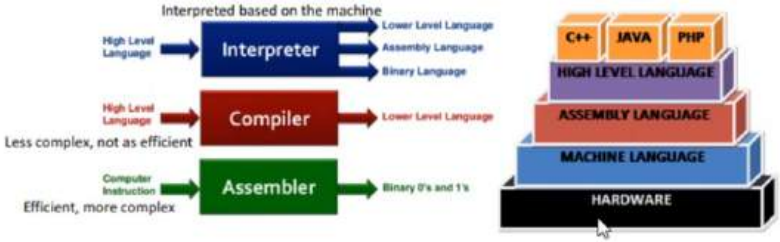
Compiler

Assemblers

Linker

Application (Binary)

Programming languages -Categories



Integrated development environment (IDE)

An integrated development environment (IDE) is a software suite that consolidates basic tools required to write and test software.

Developers use numerous tools throughout software code creation, building and testing. Development tools often include text editors, code libraries, compilers and test platforms. Without an IDE, a developer must select, deploy, integrate and manage all of these tools separately. An IDE brings many of those development-related tools together as a single framework, application or service. The integrated toolset is designed to simplify software development and can identify and minimize coding mistakes and typos.

Some IDEs are open source, while others are commercial offerings. An IDE can be a standalone application or it can be part of a larger package.

MicroPython IDE

MicroPython is an open-source programming language derived from Python 3 and designed for microcontrollers and embedded systems applications. To write and run MicroPython code we need an IDE to program the Microcontroller.

MicroPython is only a programming language interpreter and **does not include an editor**. Some MicroPython boards support a web-based code prompt/editor, but with most MicroPython boards you'll write code in your desired text editor and then use small tools to upload and run the code on a board.

MicroPython IDEs

- Mu Editor
- uPyCraft IDE
- Thonny IDE
- VS Code + Pymakr extension
- PyCharm
- microIDE

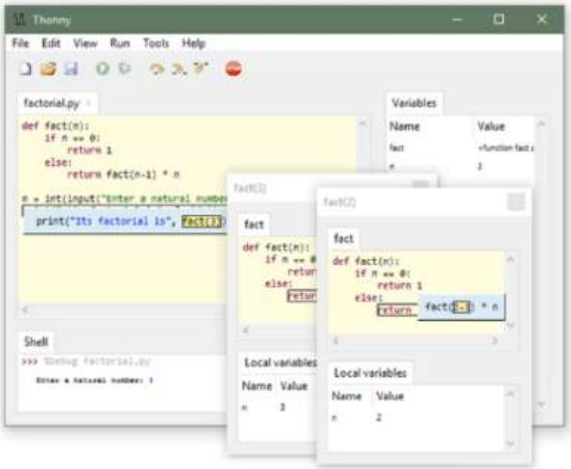
Installing MicroPython IDE

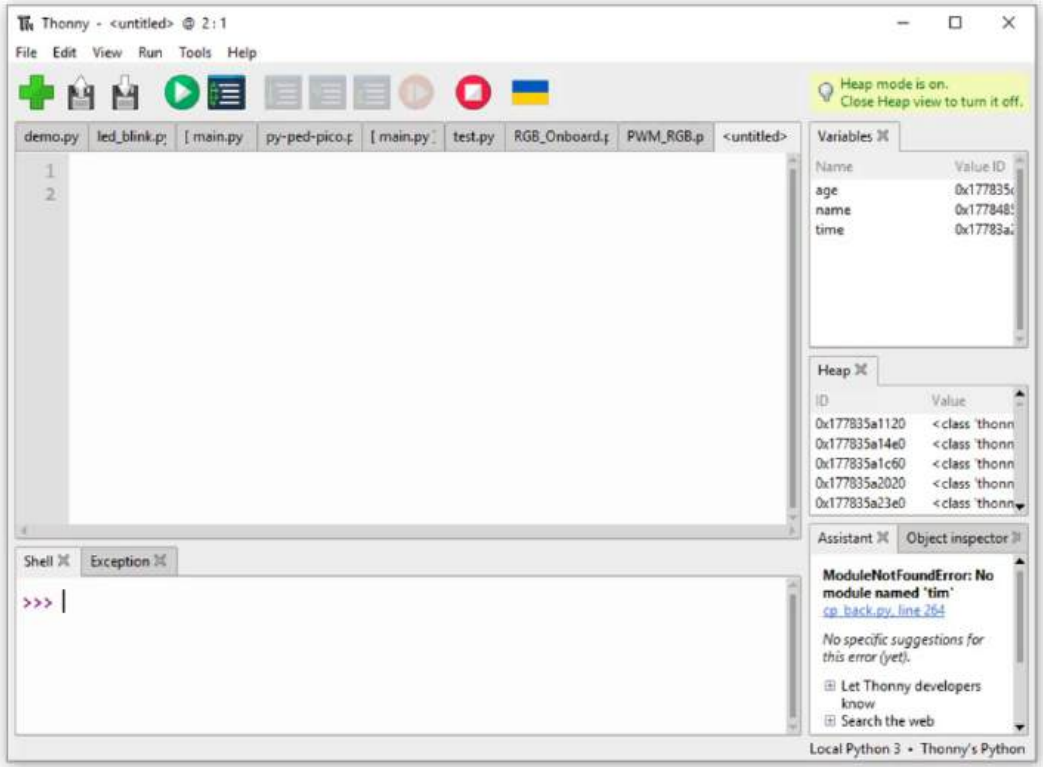
https://thonny.org/

Thonny

Python IDE for beginners

Download version 4.1.2 for Windows • Mac • Linux




Thonny - <untitled> @ 2 : 1
File Edit View Run Tools Help
Heap mode is on.
Close Heap view to turn it off.
demo.py
led_blink.p
[main.py
py-ped-pico.p
[main.py
test.py
RGB_Onboard.p
PWM_RGB.p
<untitled>
Variables
Name
Value ID
age
name
time
Heap
ID
Value
0x177835a1120
0x177835a14e0
0x177835a1c60
0x177835a2020
0x177835a23e0
Shell
Exception
>>>
Assistant
Object inspector
ModuleNotFoundError: No module named 'tim'
cp_back.py, line 264
No specific suggestions for this error (yet).
Let Thonny developers know
Search the web
Local Python 3 • Thonny's Python

MicroPython Primer

Python Variables and data Types

Manoj N
(Principal Technical Officer)
Smart Technology & Education Division

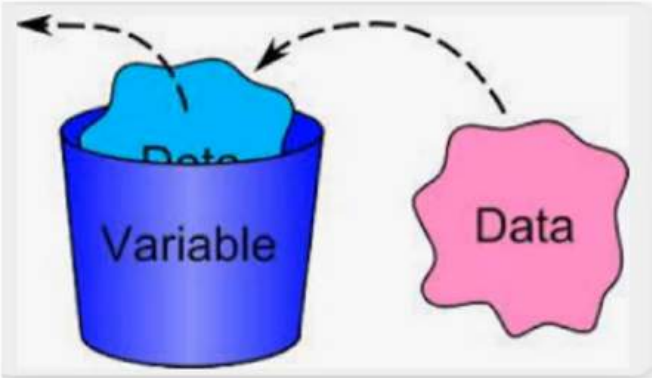
रा.इ.सू.प्रौ.सं
NIELIT

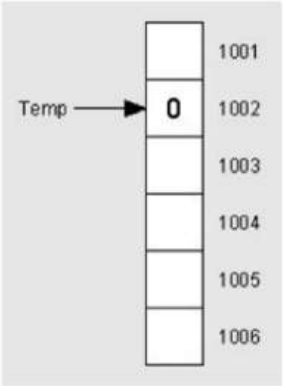
Drag from top and touch the back button to exit full screen.

NIELIT CALICUT

Variables?

A variable is a container for a value. It can be assigned a name.





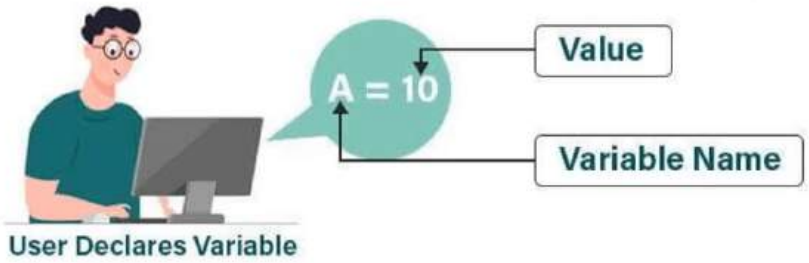
What are 'C' Variables?

```
int a = 10;
```

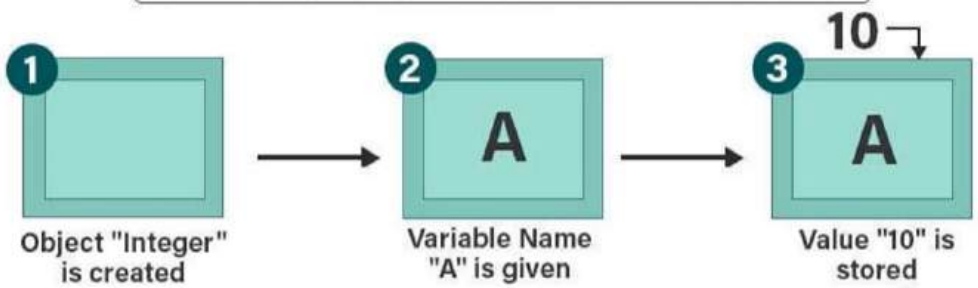
variable type — variable name — initial value

Python Variables

Python Variables are memory spaces that hold information, such as numbers or text, that the Python Code can later use to perform tasks.



WHAT HAPPENS IN PYTHON MEMORY?



Rules for naming variables

1. The variable name should start with an underscore or letter.
Example: _educba, xyz, ABC

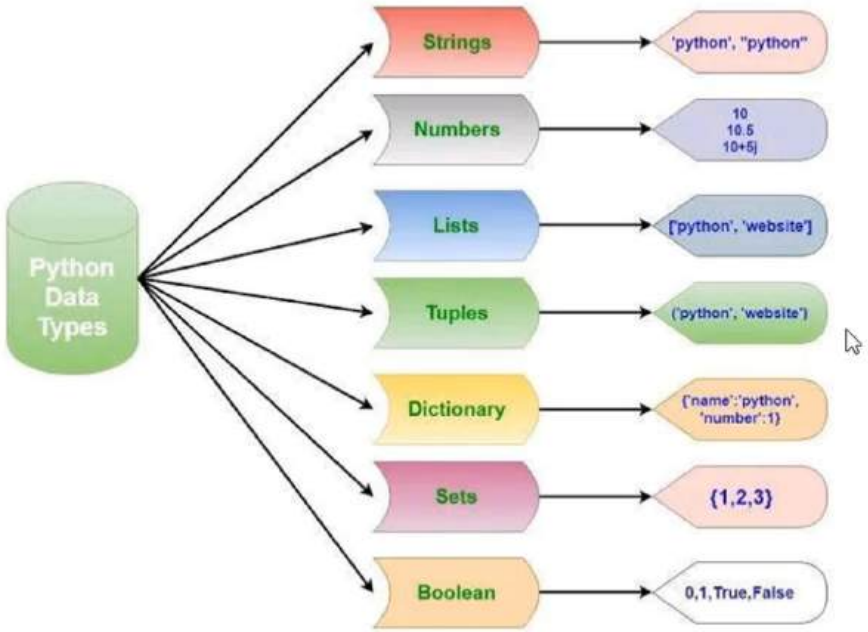
2. Using a number at the start of a variable name is not allowed.
Examples of incorrect variable names: 1variable, 23alpha, 390say

3. The variable name can only include alphanumeric characters and underscore.
Example: learn_python, c4, C564_85

4. Variable names in Python are case-sensitive.
Examples of different variable names: python, Python, PYTHON

5. Reserved words in Python cannot be a variable name.
Example: while, if, print, while

Variable data types, show the type of data stored in the variable.



Numeric Data Type in Python

The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number. These values are defined as Python int, Python float, and Python complex classes in Python.

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.

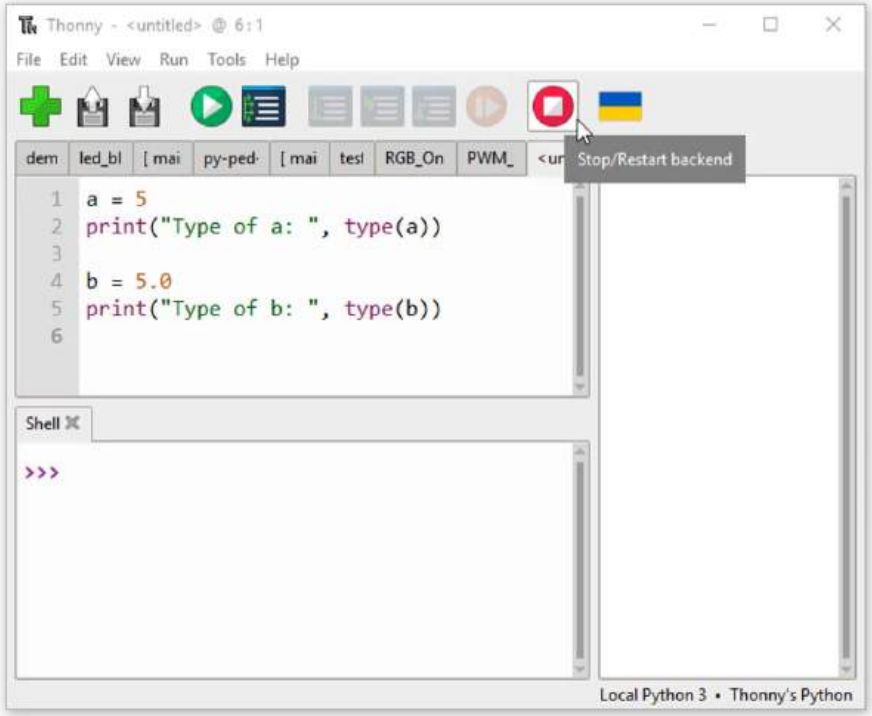
- **Float** – This value is represented by the float class. It is a real number with a floating-point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.

- **Complex Numbers** – Complex number is represented by a complex class. It is specified as *(real part) + (imaginary part)j*. For example – 2+3j

Determine the type of data type.

type() function

```
# Python program to
# demonstrate numeric value

a = 5
print("Type of a: ", type(a))

b = 5.0
print(" \nType of b: ", type(b))

c = 2 + 4j
print(" \nType of c: ", type(c))
```



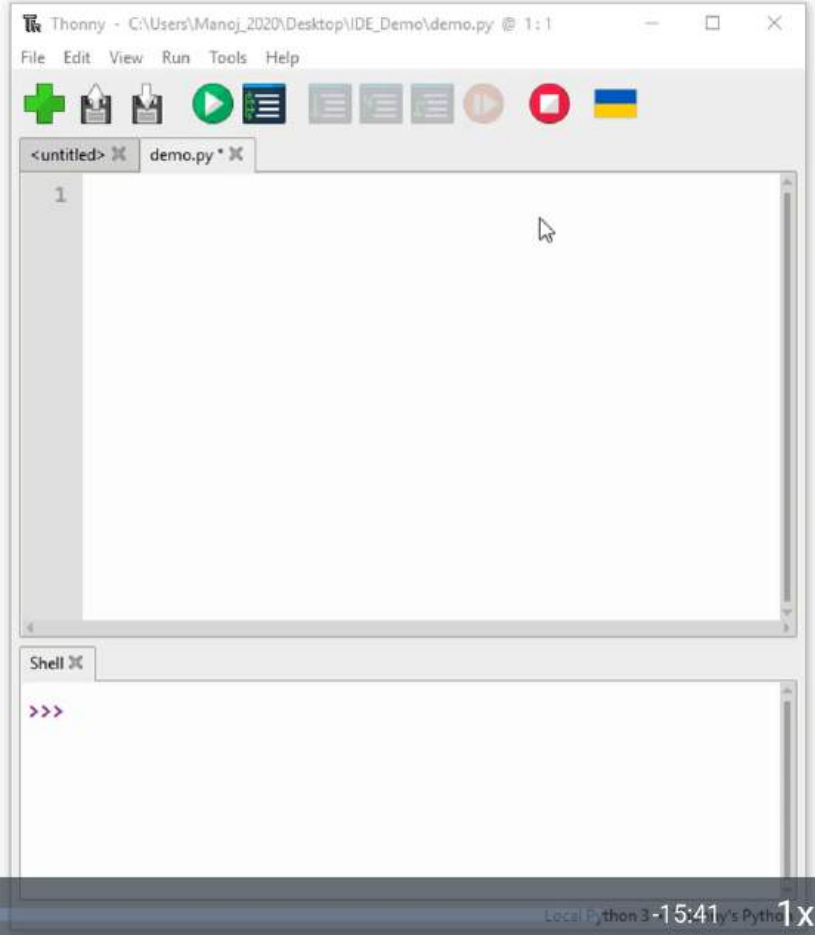
String Data Type

Strings in Python are arrays of bytes representing Unicode characters.

A string is a collection of one or more characters put in a single quote, double-quote, or triple-quote.

In python there is no character data type
A character is a string of length one.
It is represented by str class.

Creating String
Single quotes or double quotes or even triple quotes.



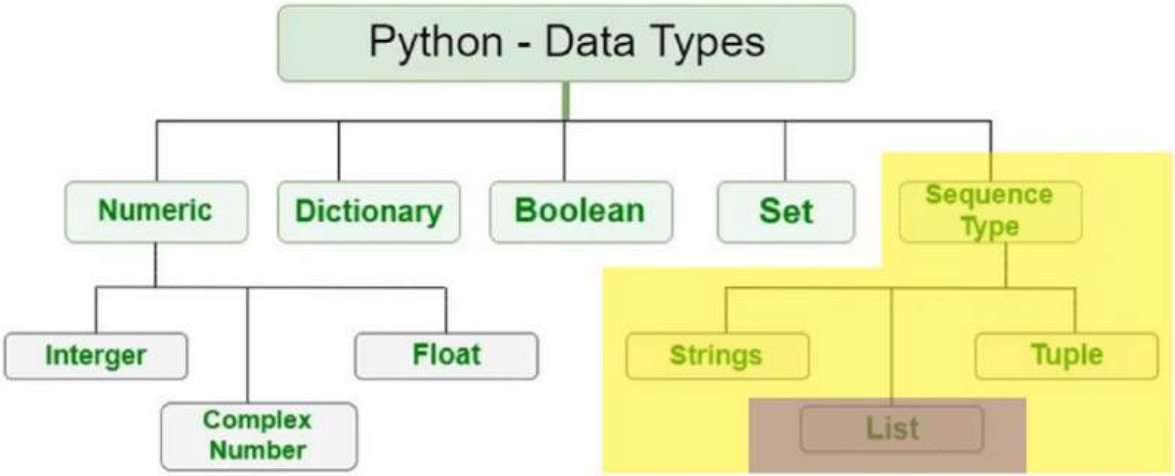
Accessing elements of String

In Python, individual characters of a String can be accessed by using the method of Indexing.
Negative Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character, and so on.

```
# Python Program to Access
# characters of String

String1 = "MICROPYTHON"
print("Initial String: ")
print(String1)

# Printing First character
print("\nFirst character of String is: ")
print(String1[0])

# Printing Last character
print("\nLast character of String is: ")
print(String1[-1])
```

M	I	C	R	O	P	Y	T	H	O	N
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Sequence Data Type in Python



List Data Type

Lists are just like arrays, declared in other languages which is an ordered collection of data.

It is very flexible as the items in a list do not need to be of the same type.

Creating List

Lists in Python can be created by just placing the sequence inside the square brackets[].

Thonny - C:\Users\Manoj_2020\Desktop\IDE_Demo\demo.py @ 5 : 15

File Edit View Run Tools Help

<untitled> demo.py *

```python
# Creating a List with the use of multiple values
List = ["Micro", "Python", "Primer"]
print("\n List containing multiple values: ")
print(List[0])
print(List[2])
print(List[-2])
```

Shell

```
>>>
```

Tuple Data Type

- Just like a list, a tuple is also an ordered collection of Python objects.
- The only difference between a tuple and a list is that tuples are immutable
- i.e. tuples cannot be modified after it is created.
- It is represented by a tuple class.

Creating a Tuple

In Python, tuples are created by placing a sequence of values separated by a 'comma' with or without the use of parentheses for grouping the data sequence.

Tuples can contain any number of elements and of any datatype (like strings, integers, lists, etc.).

Note: Tuples can also be created with a single element, but it is a bit tricky.

Having one element in the parentheses is not sufficient, there must be a trailing 'comma' to make it a tuple.

Dictionary

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered
- Changeable and
- Do not allow duplicates.

Thonny - C:\Users\Manoj_2020\Desktop\IDE_Demo\demo.py @ 7 : 1

File Edit View Run Tools Help

demo.py

```
my_dict = {
  "brand": "Tata",
  "model": "Nexon",
  "year": 2001
}
print(my_dict)
```

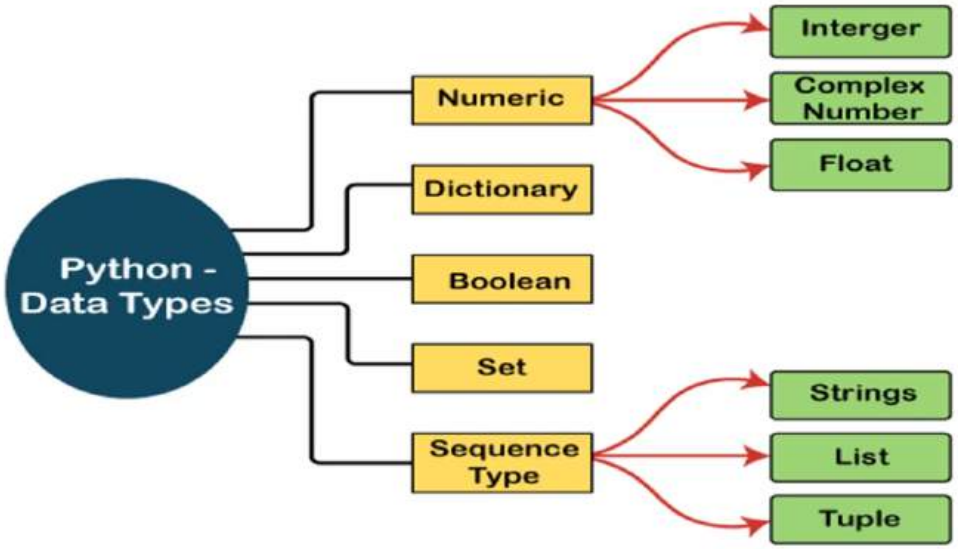
Shell

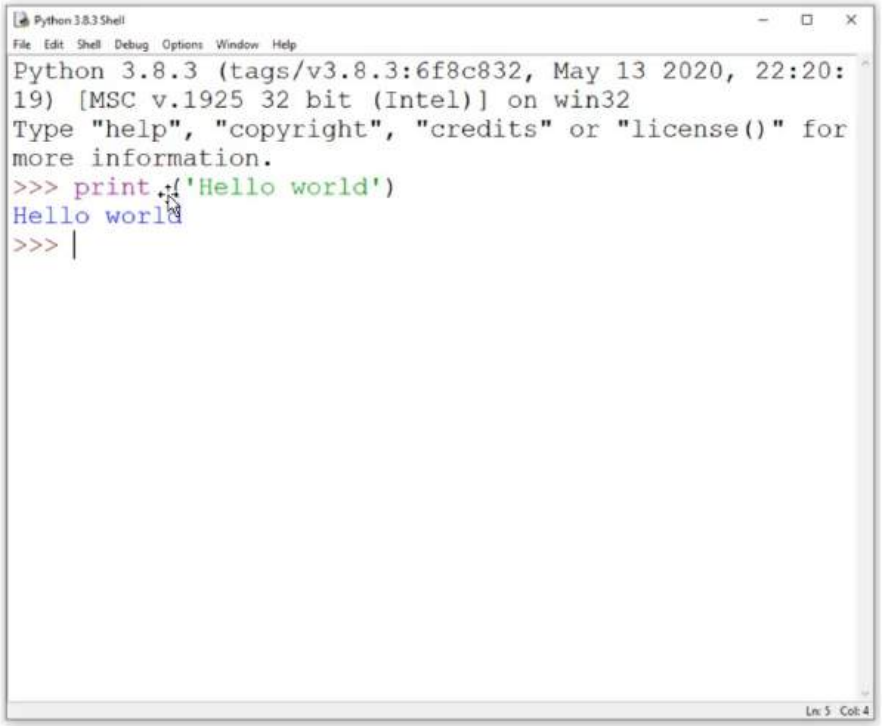
```
>>> %Run demo.py
 {'brand': 'Tata', 'model': 'Nexon', 'year': 2001}
>>>
```

Local Python 3 • Thonny's Python

Python Booleans

Booleans represent one of two values: True or False.


Python - Data Types
Numeric
Dictionary
Boolean
Set
Sequence Type
Interger
Complex Number
Float
Strings
List
Tuple


Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:
19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for
more information.
>>> print ('Hello world')
Hello world
>>>
Ln: 5 Col: 4

-24:09 1x

रा.इ.सू.प्रौ.सं NIELIT | राष्ट्रीय इलेक्ट्रॉनिकी एवं सूचना प्रौद्योगिकी संस्थान
National Institute of Electronics & Information Technology

Ministry of Electronics & Information Technology
Government of India

Python Functions

RAJESH M.

Agenda

- Overview of function
- Function syntax
- Passing arguments and return
- Built in functions
- Best practices

Overview of function

- Definition: A function is a block of code that performs a specific task.
- Functions are essential for code organization and reusability.
- Functions make your code more modular and easier to maintain.
- Example: print(“hello”)

Function Syntax in Python

- In Python a function is defined using the `def` keyword

Syntax: def function_name(parameters):

Example : test.py

```
def my_function():
  print("Hello from a function")


my_function()
```

Passing arguments and Return values in functions

Example : test.py

```
def my_sum(x,y):
  s=x+y
  return s


result=my_sum(10,20)
print(result)
```

Passing arguments and Return values in functions

Example : sum_alternate.py

```
def my_sum(x,y):
  s=x+y
  return s


result=my_sum(x=10,y=20)
print(result)
```

Arbitrary arguments

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a *tuple* of arguments, and can access the items accordingly

```
def my_function(*centre):

  print("The NIELIT Centre  " + centre[0])


my_function("Calicut", "Chennai", "Aurangabad", "New Delhi", "Gorakhpur", "Chandigarh")
```

Scope of function

- Scope: The region where a variable is accessible.
- Local variables: Defined within a function, limited to that function.
- Global variables: Defined outside functions, accessible everywhere.

Lambda function

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.

Syntax

```
lambda arguments : expression
```

- The expression is executed and the result is returned:

```
x = lambda a : a + 10

print(x(5))

#output = 15
```

Best practices

- Encourage good naming conventions for functions.
- Keep functions small and focused on a single task.
- Use comments and docstrings for documentation.
- Highlight the importance of testing functions.

रा.इ.सू.प्रौ.सं NIELIT | राष्ट्रीय इलेक्ट्रॉनिकी एवं सूचना प्रौद्योगिकी संस्थान
National Institute of Electronics & Information Technology

Ministry of Electronics & Information Technology
Government of India

Python Classes and Objects

RAJESH M.

Overview of class and objects

- Python is an object oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.
  - Objects: Instances of classes that encapsulate data and behavior.
  - Object-Oriented Programming (OOP): A paradigm based on classes and objects.

Creating Class in Python

- To create a class, use the keyword `class`: keyword

Syntax: class class_name :

Example : test.py

```
class MyClass:
  x = 5      #x is an datameber in the class
p1=Myclass()    #create p1 as an object of Myclass  or instantiate object from an class
print(p1.x)
```

Constructor

- __init__ method: Special method used for object initialization.
- Pass initial data to objects during creation.
- Example:

```
class MyClass:
    def __init__(self, name):   #member function
        # Public member
        self.name = name        #data member
```

```
P1=Myclass("NIELIT");
print(P1.name)
```

Self parameter

- The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
- It does not have to be named `self` , you can call it whatever you like, but it has to be the first parameter of any function in the class:

```
class MyClass:
  def __init__(self, name):   #member function
    # Public member
    self.name = name      #data member

P1=Myclass("NIELIT");
print(P1.name)
```

```
class MyClass:
  def __init__(myself, name):   #member function
    # Public member
    myself.name = name      #data member

P1=Myclass("NIELIT");
print(P1.name)
```

Private and public

```
class MyClass:
  def __init__(self, name):
    # Public member
    self.name = name
    # Private member (name mangling)
    self.__age = 0
    # Public method to set the age
  def set_age(self, age):
    if age >= 0:
      self.__age = age
# Public method to get the age
  def get_age(self):
    return self.__age
  def display_info(self):
    print(f"Name: {self.name}")
    print(f"Age: {self.__age}")
```

```
# Create an instance of MyClass
obj = MyClass("John")
# Access public member 'name'
print(obj.name)  # Output: John
# Access private member 'age' using a public method
obj.set_age(30)
# Access private member 'age' using a public method
print(obj.get_age())  # Output: 30

# Access private member '__age' directly (name mangling)
# Note that this is possible but discouraged
print(obj._MyClass__age)  # Output: 30

# Call a public method to display information
obj.display_info()
```

Python Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class.

```
#parent Class
class Person:
  def __init__(self, fname,
lname):
    self.firstname = fname
     self.lastname = lname

  def printname(self):
     print(self.firstname,
self.lastname)

#Use the Person class to create
an object, and then execute the
printname method:

x = Person("Vijay", "Rahul")
x.printname()
```

```
Child
class Student(Person):
  Pass

x = Student("Jio", "Virat")
x.printname()
```

Use the `pass` keyword when you do not want to add any other properties or methods to the class.

Child overrides parent

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    self.firstname = lname
    self.lastname = fname
   # Person.__init__(self, fname, lname)

x = Student("Rahul", "Bharat")
x.printname()
```

The child's `__init__()` function overrides the inheritance of the parent's `__init__()` function.

Super() function

- By using the `super()` function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.
- Python also has a `super()` function that will make the child class inherit all the methods and properties from its parent:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)
```

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

x = Student("Arun", "Maya", 2019)
print(x.graduationyear)
print(x.printname())
```

Python Polymorphism

- The word "polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes. Example shows len() usage in three different forms.

```
x = "Hello World!"
y = ("apple", "banana", "cherry")
z = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

print(len(x))
print(len(y))
print(len(z))
```

Output

```
12
3
3
```

Best practices

- Use clear and meaningful class and method names.
- Follow naming conventions.
- Document classes and methods using docstrings.
- Encourage code reusability through inheritance.

Agenda

- Overview of Python Modules
- Creating Modules
- Import modules
- Built in modules
- Best Practices

रा.इ.सू.प्रौ.सं NIELIT | राष्ट्रीय इलेक्ट्रॉनिकी एवं सूचना प्रौद्योगिकी संस्थान
National Institute of Electronics & Information Technology

Ministry of Electronics & Information Technology
Government of India

Python Modules

RAJESH M

Agenda

- Overview of Python Modules
- Creating Modules
- Import modules
- Built in modules
- Best Practices

Python Modules

- Definition: A module is a Python script that contains reusable code.
- Modules help in organizing code and making it more maintainable.
- Python's extensive standard library is organized into modules.

Creating a Python Module

- Use any Python script as a module.
- Save a Python script with a .py extension (e.g., test_module.py).
- You can define functions, classes, and variables within the module.
- Example: centre_module.py

```
def greeting(name):

  print("Welcome to , " + name)

details= {

  "City": "Calicut",

  "State": "Kerala",

  "country": "India"

}
```

Application of Python Modules

- Code Reusability
- Maintainability.
- Collaboration.
- Helps to Reduce Namespace Conflicts.

Importing Modules

- import statement: Used to bring a module into your code.
- Syntax: import module_name
- Usage - Using import statement we can access module:

Example test.py

```
import centre_module

centre_module.greeting("NIELIT")
```

Importing a specific member from Module

- Importing Specific Members
- Import specific functions or variables using from.
- Syntax: from module_name import member
- Example: test2.py

```
import centre_module

from centre_module  import details

     centre_module.greeting("NIELIT")

     print(details["city"])
```

Aliasing Modules

- You can create an alias / rename when you import a module, by using the as keyword:

```
import centre_module as centre

    centre.greeting("NIELIT")
```

Built in Modules

- Python offers a rich standard library with many built-in modules.
- Examples: platform, math, random, datetime, os, etc.

Using dir() function

- There is a built-in function to list all the function names (or variable names) in a module.

```
import math

x = dir(math)

print(x)
```

Best practices

- Use descriptive module names.
- Organize related functions and classes in a module.
- Avoid global variables in modules.
- Use comments and docstrings for documentation.

रा.इ.सू.प्रौ.सं NIELIT | राष्ट्रीय इलेक्ट्रॉनिकी एवं सूचना प्रौद्योगिकी संस्थान
National Institute of Electronics & Information Technology

Ministry of Electronics & Information Technology
Government of India

Overview of Microcontroller for Embedded Systems

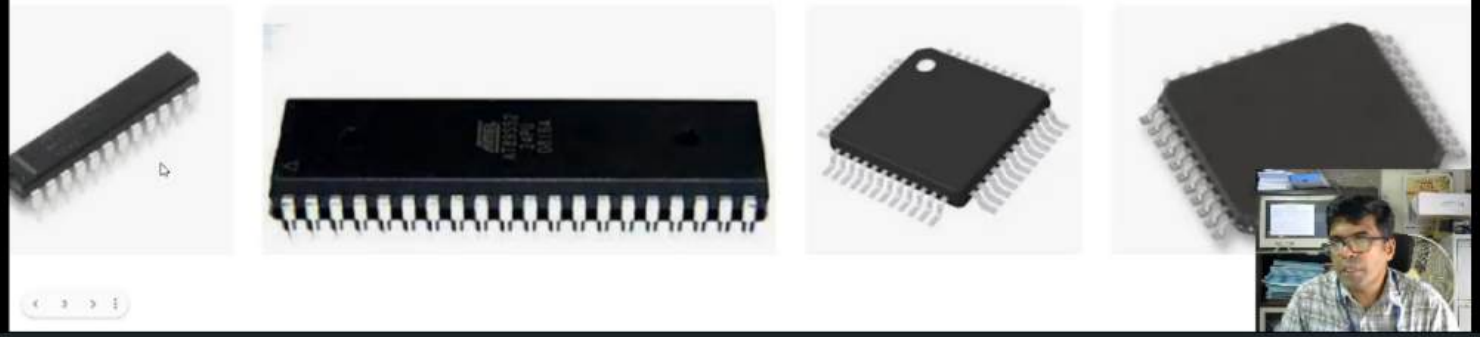
RAJESH M

Agenda

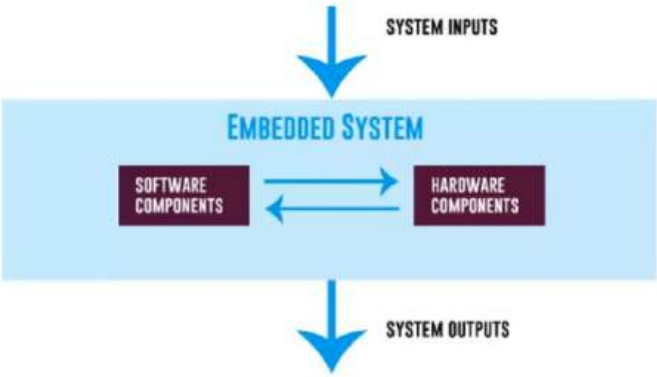
- Overview of Microcontrollers
- Overview Embedded Systems
- Architecture of Microcontrollers
- Microcontroller Selection criteria
- Integrated Development Environment
- Best Practices

Overview of Microcontrollers

- Microcontrollers are small, integrated computing devices.
- Purpose: Control and monitor various hardware components in embedded systems.

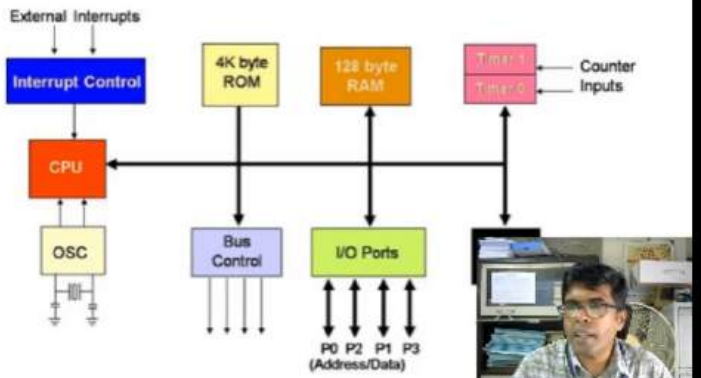


Embedded Systems



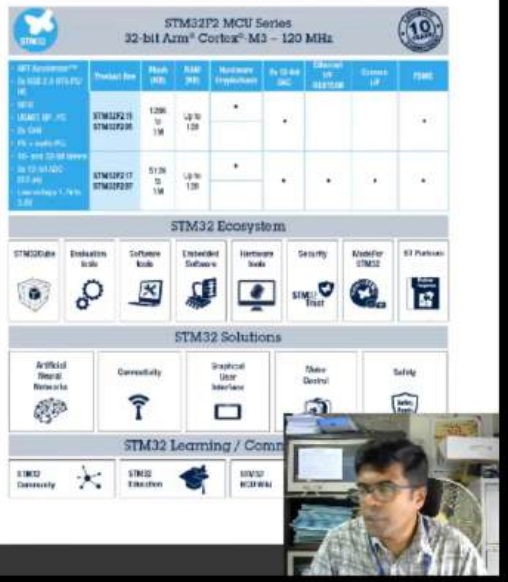
Architecture of a typical Microcontroller

- CPU: Central Processing Unit.
- Memory: ROM (Program), RAM (Data), Flash (Storage).
- Input/Output: Ports for sensors, displays, etc.
- Peripherals: Timers, counters, UART, SPI, I2C, etc.
- Clock Source: Determines execution speed.

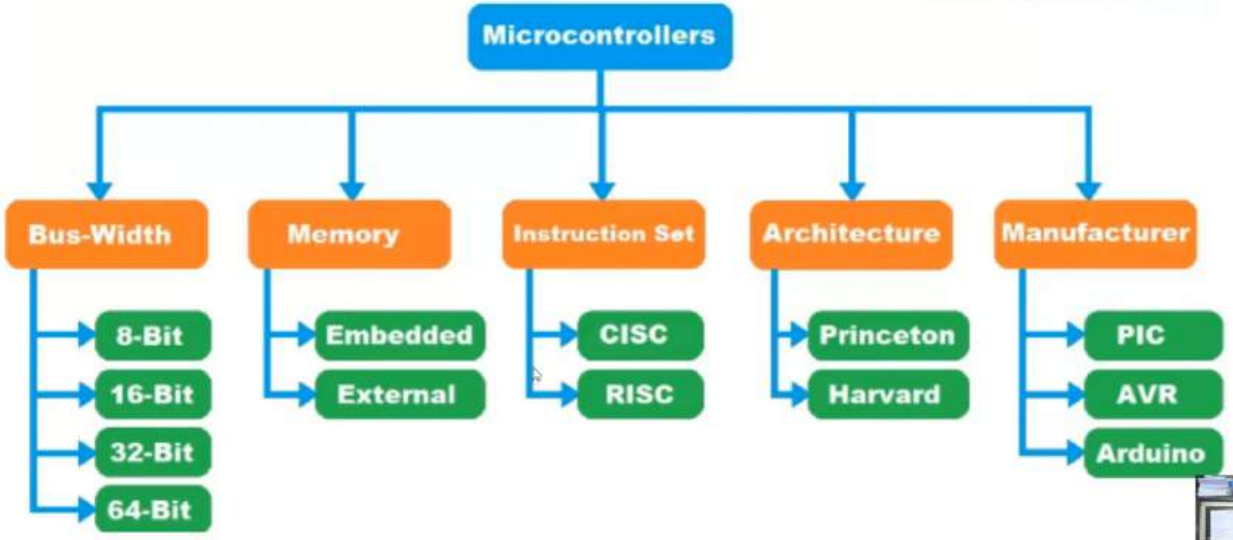


Key characteristics

- Size: Compact, low-power devices.
- Peripherals: Input and output capabilities.
- On-chip Memory: ROM, RAM, and Flash.
- Real-time Operation: Precise timing and control.
- Low Cost: Designed for mass production.



Types of Microcontroller



Microcontroller Vs Microprocessor Vs Soc

Application Areas

- Automotive: Engine control, infotainment systems.
- Consumer Electronics: Smartphones, appliances.
- Industrial Automation: Robotics.
- Medical Devices: Monitoring and control.
- IoT Devices: Sensors, actuators, connectivity.

Microcontroller Selections

- Performance: Match processing power to the task.
- Peripherals: Availability and compatibility.
- Power Consumption: Critical for battery-powered devices.
- Memory: Sufficient storage for program and data.
- Development Tools: Availability of IDEs, compilers, simulators.

Development Environment

- IDEs: Integrated Development Environments.
- Compilers: Translate high-level code to machine code.
- Debugging Tools: For testing and troubleshooting.
- Simulation: Test code without hardware.

Microcontroller Programing flow - Generic

Microcontroller's Compiler

It **Compiles** the code

& **Generates** the .HEX File

Send **.Hex File** to Microcontroller's

Programmer/Burner (i.e. PICkit3)

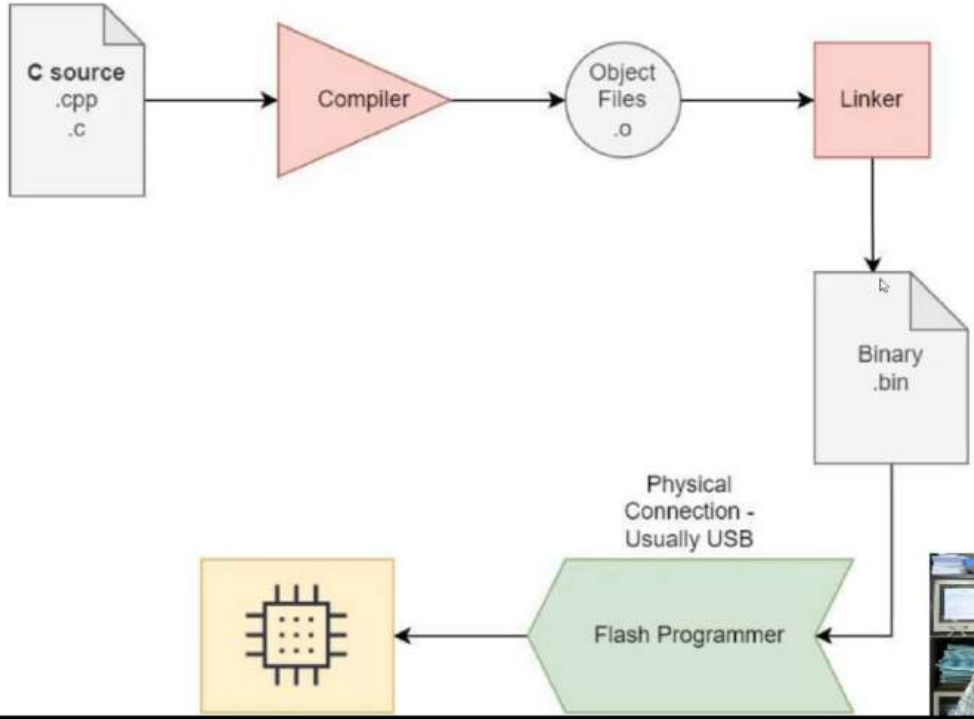
Microcontroller's Programmer

It **Uploads** the .Hex File in

Microcontroller's **ROM**

Flow Diagram for Programming a Microcontroller

Microcontroller Programing flow - Generic



Creating Challenges

- Limited Resources: Memory, processing power.
- Real-time Constraints: Meet strict timing requirements.
- Power Management: Optimize for low power.
- Compatibility: Ensure compatibility with other components.
- Security: Protect against vulnerabilities.

Best practices

- Code Optimization: Write efficient code.
- Documentation: Clearly document code and hardware.
- Testing: Rigorous testing and validation.
- Version Control: Use version control systems.
- Security: Implement security measures.

MicroPython Primer

Raspberry Pi Pico W

Manoj N
(Principal Technical Officer)
Smart Technology & Education Division

रा.इ.सू.प्रौ.सं
NIELIT

NIELIT CALICUT

This explains how to use MicroPython programming language,

What is MicroPython?

MicroPython is a tiny open source Python programming language interpretor that runs on small embedded development boards. With MicroPython you can write clean and simple Python code to control hardware instead of having to use complex low-level languages like C or C++

The simplicity of the Python programming language makes MicroPython an excellent choice for beginners who are new to programming and hardware.

However MicroPython is also quite full-featured and supports most of Python's syntax

Using Raspberry Pi Pico W for Embedded Systems

-6:37 1x

Raspberry Pi Pico development boards

RPi Pico

RPi Pico H

RPi Pico W

RPi Pico WH

Using Raspberry Pi Pico W for Embedded Systems

- IO Interfaces
  - Digital I/O, PWM
- Communication Protocols
  - UART, I2C, SPI
- On Chip Peripherals
  - ADC, Temperature Sensors

What is Raspberry pi pico w?

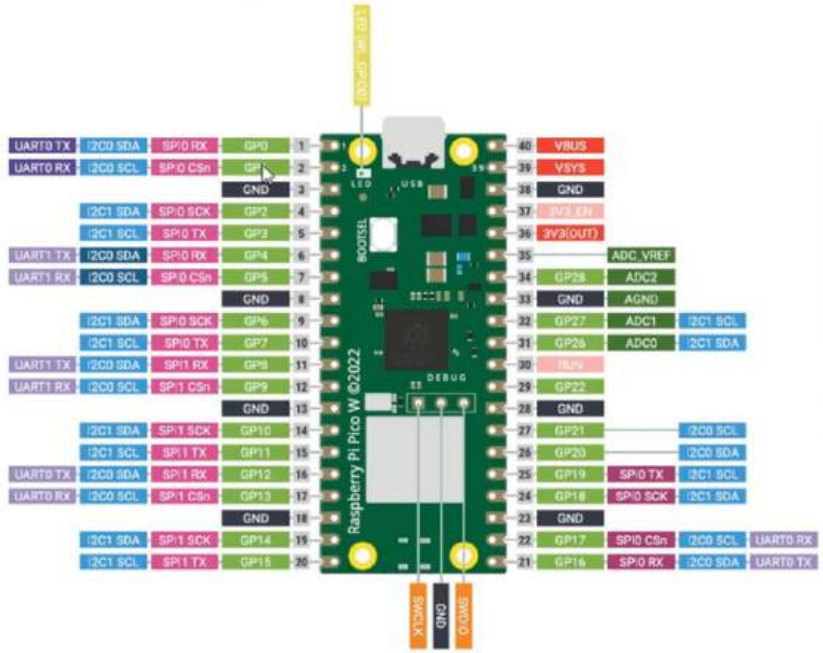
What is PICO andPICO W

- PICO and PICO W are the boards designed to use RP2040 by raspberry pi foundation
- You can design your own pico board as well
- Difference being PICO W comes with a wireless interface (wifi)
- Suitable for general purpose embedded system projects and IoT
- Can be used with Thonny Python IDE with Micropython

RP2040

- RP2040 microcontroller chip designed by Raspberry Pi in the UK
- Dual-core Arm Cortex-M0+ processor, flexible clock running up to 133 MHz
- 264kB on-chip SRAM, 2MB on-board QSPI flash
- 2.4GHz 802.11n wireless LAN (Raspberry Pi Pico W and WH only)
- 26 multifunction GPIO pins, including 3 analog inputs
- 2 × UART, 2 × SPI controllers, 2 × I2C controllers, 16 × PWM channels
- 1 × USB 1.1 controller and PHY, with host and device support
- Supported input power 1.8–5.5V DC
- Accurate on-chip clock
- Temperature sensor

Raspberry Pi PICO-W Pinout



Raspberry Pi Documentation -

https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html

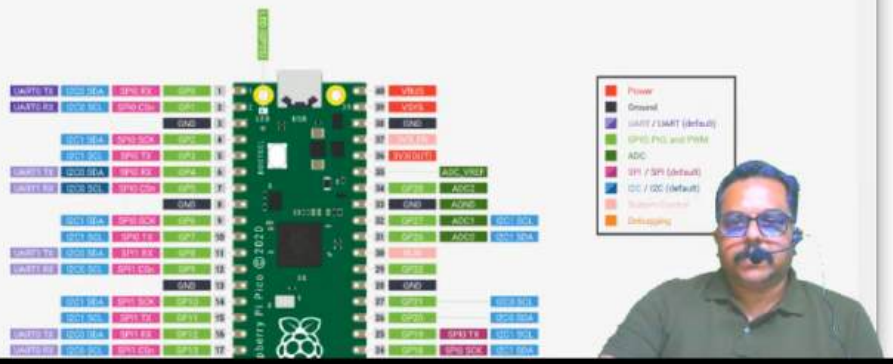
110%

boards, while the Pico H comes with pre-soldered headers.

NOTE

Both boards have a three pin Serial Wire Debug (SWD) header. However, the Pico H has this broken out into a small, keyed, 3-pin connector while the Pico has three castellated through-hole pins adjacent to the edge of the board.

Pinout and design files



Install Firmware

- Connect usb cable to Raspberry Pi Pico, not to PC
- Press Bootsel button
- Now connect other end of cable to PC
- Raspberry Pi opens as a storage device
- Open the https://micropython.org/download/rp2-pico-w/
- Download uf2 file
- Save it to storage device Done!!!

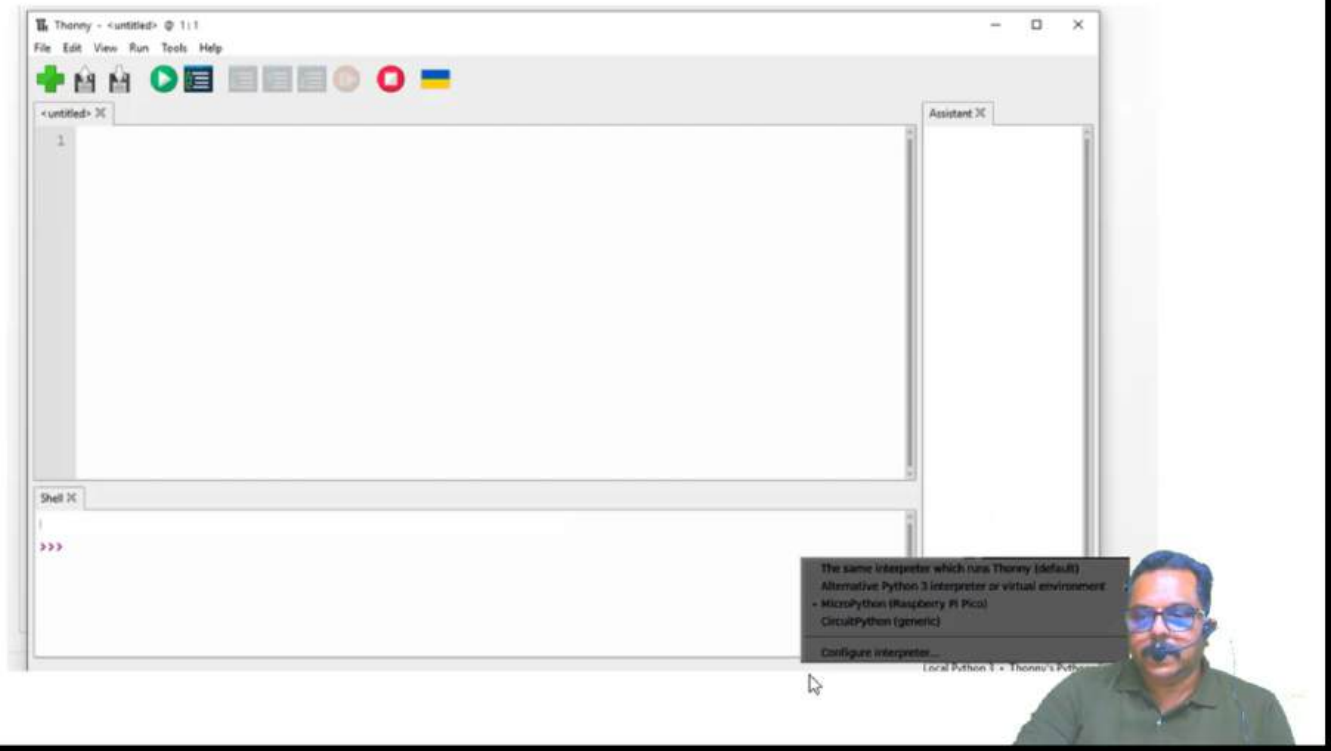
MicroPython Primer

Application Development using MicroPython on Raspberry Pi Pico W

Manoj N
(Principal Technical Officer)
Smart Technology & Education Division

रा.इ.सू.प्रौ.सं
NIELIT

NIELIT CALICUT


Thonny - <untitled> @ 1 : 1
File Edit View Run Tools Help
<untitled>
Assistant
Shell
>>>
The same interpreter which runs Thonny (default)
Alternative Python 3 interpreter or virtual environment
MicroPython (Raspberry Pi Pico)
CircuitPython (generic)
Configure interpreter...

LED Blink on Raspberry Pi pico W on boot

Using Micro Python

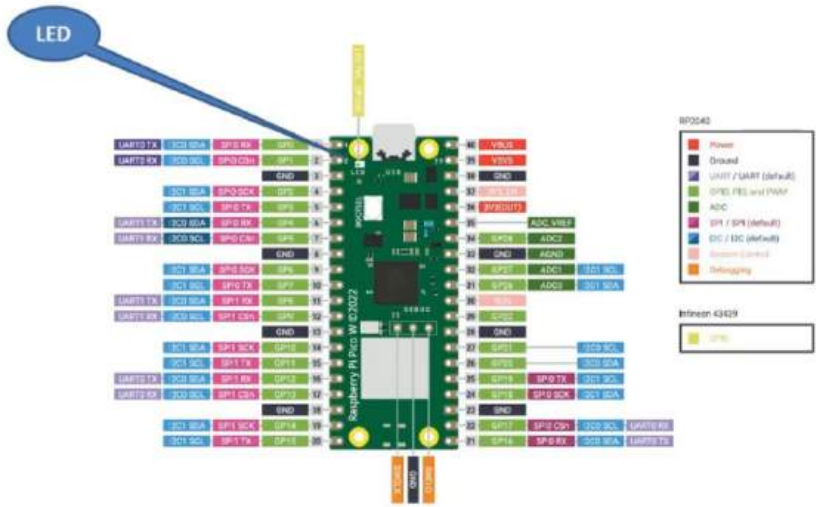
Turn on LED PICO W

Onchip LED on PICO is connected at GPIO25, ***but*** on PICO W, its connected to a wireless controller pin, referred to as "LED" in micropython

Turn on LED PICO W

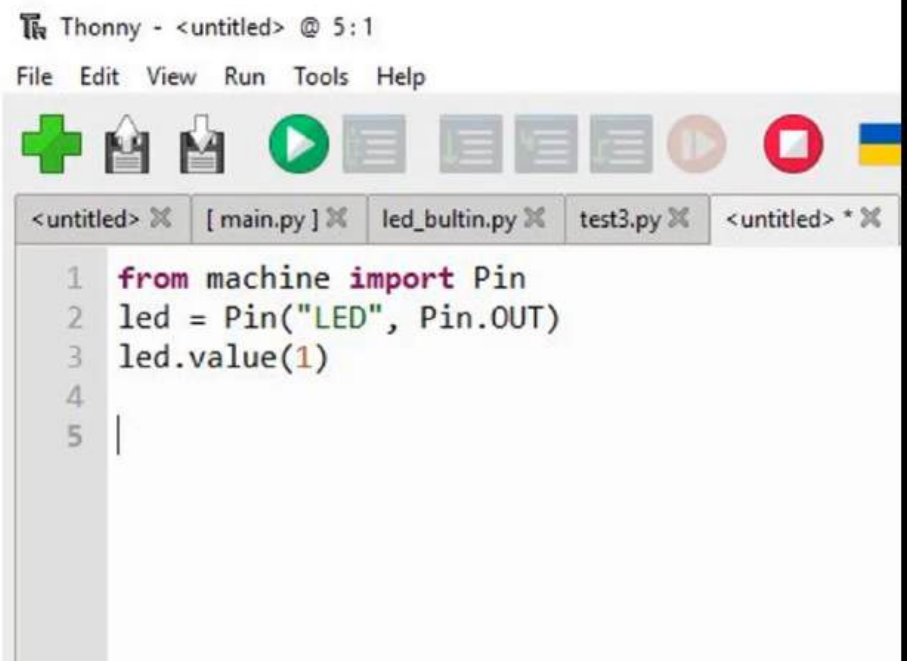
Onchip LED on PICO is connected at GPIO25, ***but*** on PICO W, its connected to a wireless controller pin, referred to as "LED" in micropython

Raspberry Pi PICO-W Pinout



Turn on LED PICO W

Onchip LED on PICO is connected at GPIO25, ***but*** on PICO W, its connected to a wireless controller pin, referred to as "LED" in micropython



Thonny - <untitled> @ 5 : 1

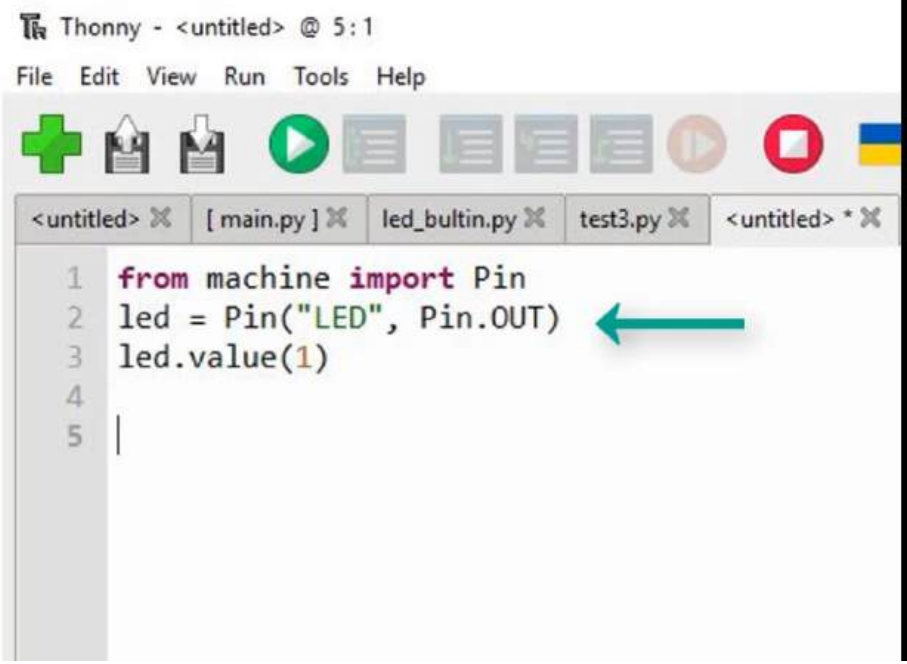
File Edit View Run Tools Help

<untitled> | [main.py] | led_bultin.py | test3.py | <untitled> *

```
from machine import Pin
led = Pin("LED", Pin.OUT)
led.value(1)

```

Turn on LED PICO W

Onchip LED on PICO is connected at GPIO25, ***but*** on PICO W, its connected to a wireless controller pin, referred to as "LED" in micropython



Thonny - Raspberry Pi Pico :: /led.py @ 10 : 17

File Edit View Run Tools Help

<untitled> | [main.py] | led_builtin.py | test3.py | <untitled> | [led.py]

```
#program to toggle the LED @ 25
import time
from machine import Pin
led = Pin("LED", Pin.OUT)
while True:

    led.value(1)
    time.sleep(2)
    led.value(0)
    time.sleep(2)
```

Shell

```
>>> %Run -c $EDITOR_CONTENT
```

MicroPython (Raspberry Pi Pico) • COM217

MicroPython Primer

Micropython Programming- PWM

Manoj N
(Principal Technical Officer)
Smart Technology & Education Division

रा.इ.सू.प्रौ.सं
NIELIT

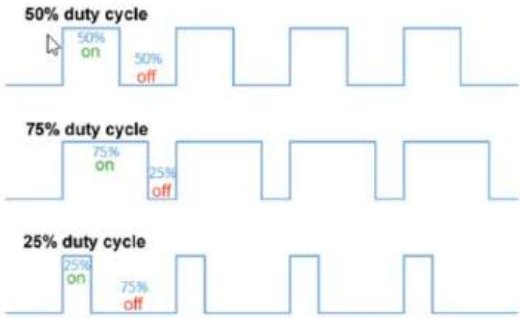
NIELIT CALICUT

PWM on RPi PICO W

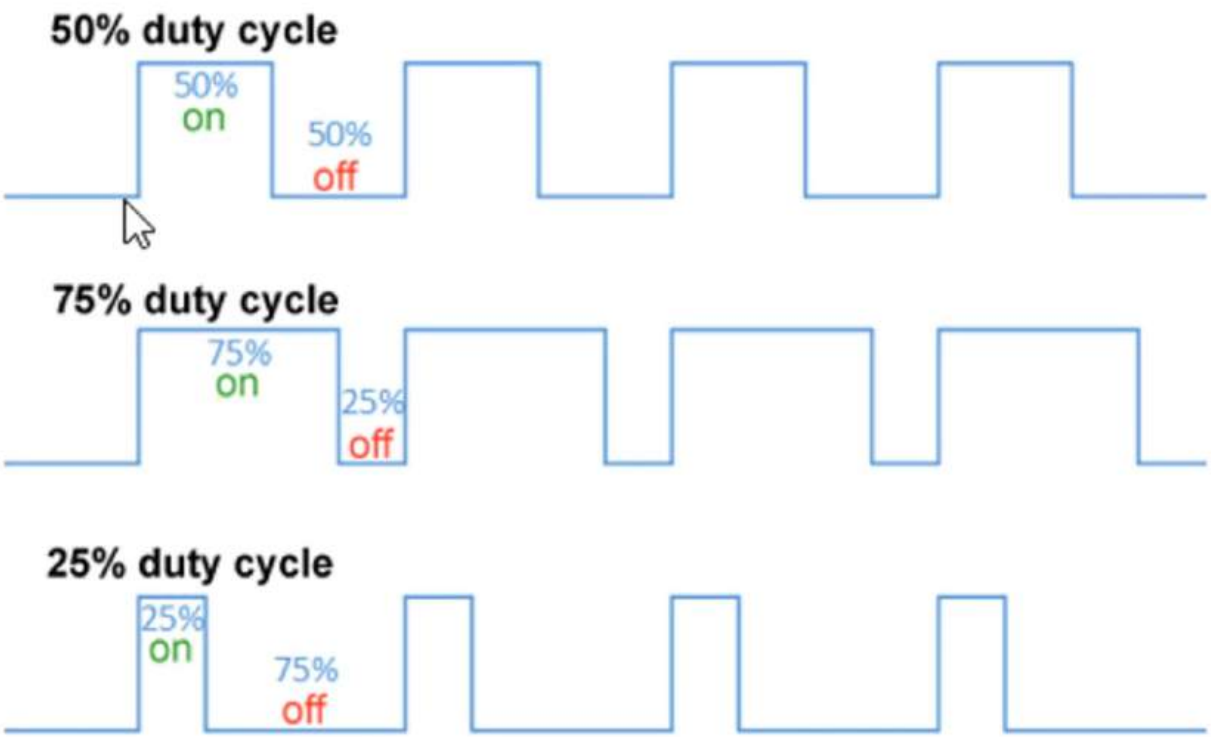
Using Micro Python

PWM - Concept

- Pulse width modulation (PWM) is a scheme where a digital signal provides a smoothly varying average voltage.
- This is achieved with positive pulses of some controlled width, at regular intervals.
- The fraction of time spent high is known as the duty cycle.
- This may be used to approximate an analog output, or control switchmode power electronics



How to use PWM on Raspberry Pi PICO W


50% duty cycle
50%
on
50%
off
75% duty cycle
75%
on
25%
off
25% duty cycle
25%
on
75%
off

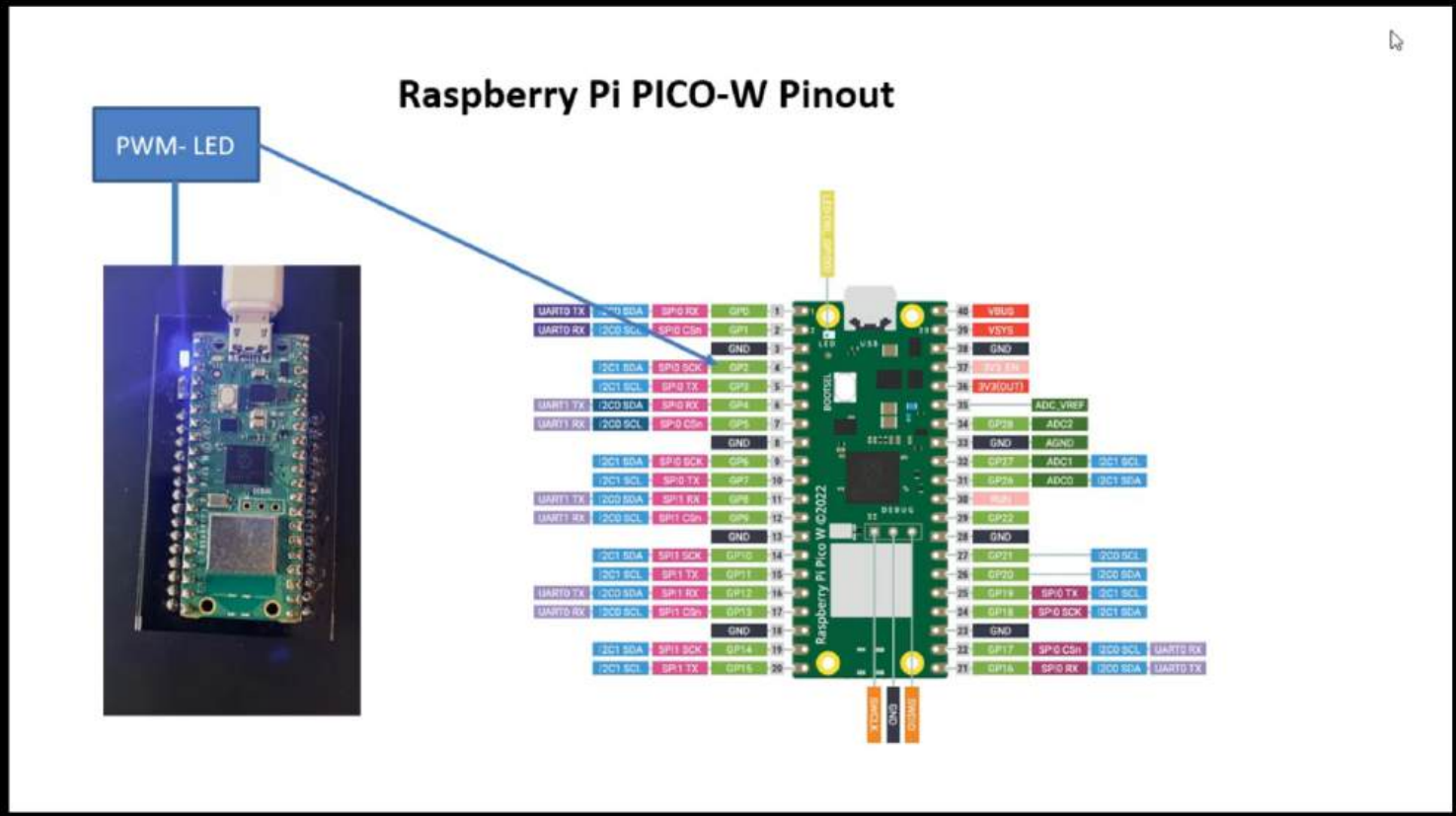
-5:52 1x

Applications of PWM

- Controlled Voltage output – LEDs Fading
- Buzzer Sound Generation
- Firing Angle Control – Lamp Dimming and several such usages
- Speed Control of DC Motor

PWM Output on PICO W

- RP2040 has 8 PWM “Slices” (or modules)
- Each Slice has 2 channels (A / B)
- Both can be used as output, or One can be used as Input
- Total 16 PWM outputs
- Any Pin of RP2040 can be configured as PWM Pin



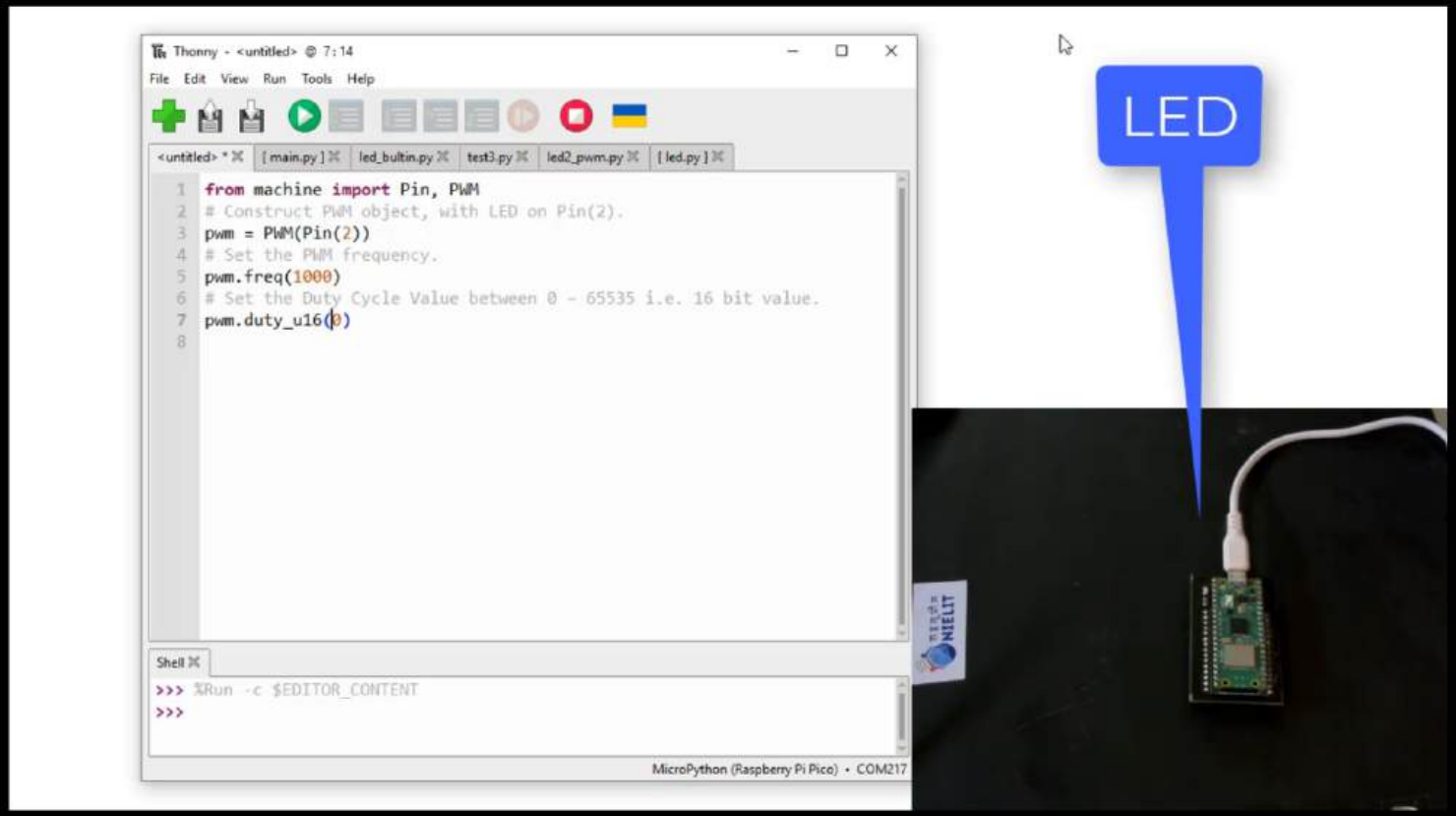
-4:19 1x

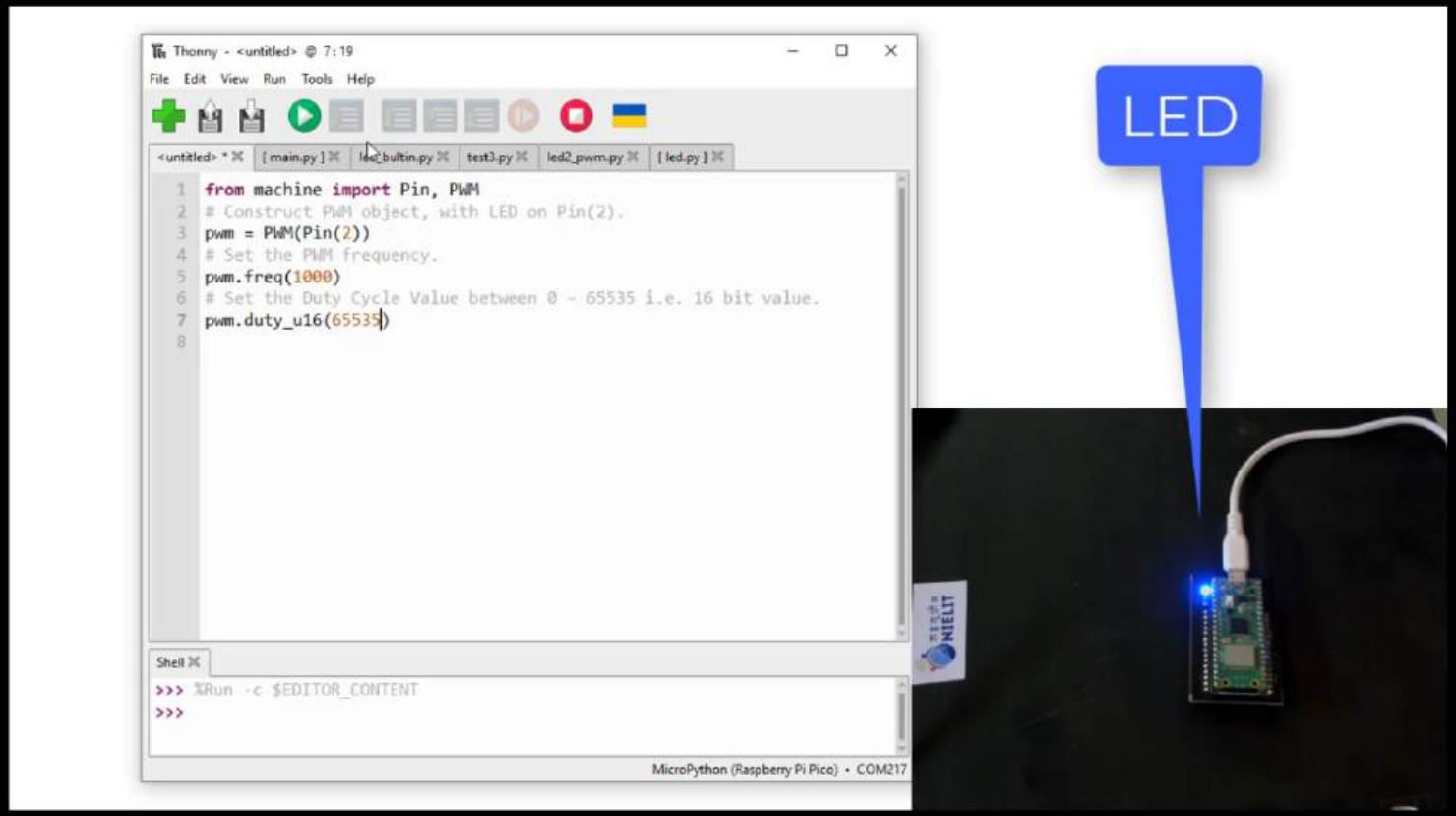
Thonny - <untitled> @ 7:14

File Edit View Run Tools Help

<untitled> * | [main.py] | led_bultin.py | test3.py | led2_pwm.py | [led.py]

```
from machine import Pin, PWM
# Construct PWM object, with LED on Pin(2).
pwm = PWM(Pin(2))
# Set the PWM frequency.
pwm.freq(1000)
# Set the Duty Cycle Value between 0 - 65535 i.e. 16 bit value.
pwm.duty_u16(0)
```


Thonny - <untitled> @ 7:14
File Edit View Run Tools Help
<untitled> * [main.py] led_bultin.py test3.py led2_pwm.py [led.py]
from machine import Pin, PWM
# Construct PWM object, with LED on Pin(2).
pwm = PWM(Pin(2))
# Set the PWM frequency.
pwm.freq(1000)
# Set the Duty Cycle Value between 0 - 65535 i.e. 16 bit value.
pwm.duty_u16(0)
Shell
>>> %Run -c $EDITOR_CONTENT
>>>
MicroPython (Raspberry Pi Pico) • COM217
LED


Thonny - <untitled> @ 7:19
File Edit View Run Tools Help
<untitled> * main.py led_builtin.py test3.py led2_pwm.py led.py
from machine import Pin, PWM
# Construct PWM object, with LED on Pin(2).
pwm = PWM(Pin(2))
# Set the PWM frequency.
pwm.freq(1000)
# Set the Duty Cycle Value between 0 - 65535 i.e. 16 bit value.
pwm.duty_u16(65535)
Shell
>>> %Run -c $EDITOR_CONTENT
>>>
MicroPython (Raspberry Pi Pico) • COM217
LED

Thonny - C:\Users\Manoj_2020\Documents\led2_pwm.py @ 9 : 1

File Edit View Run Tools Help

<untitled> * | [main.py] | led_bultin.py | test3.py | led2_pwm.py * | [led.py]

```
# Construct PWM object, with LED on Pin(2).
pwm1 = PWM(Pin(2))
# Set the PWM frequency.
pwm1.freq(1000)
# Set the Duty Cycle Value between 0 - 65535 i.e. 16 bit value.

while True:
    for i in range(0,65535,50):
        pwm1.duty_u16(i)
        time.sleep(0.001)

    print("100% ON")
    time.sleep(2)

    for i in range(65535, 0, -50):
        pwm1.duty_u16(i)
        time.sleep(0.001)

    print("100% OFF")
    time.sleep(2)
```

Shell

```
>>> %Run -c $EDITOR_CONTENT
>>>
```

MicroPython (Raspberry Pi Pico) • COM217

Thonny - C:\Users\Manoj_2020\Documents\led2_pwm.py @ 9 : 1

File Edit View Run Tools Help

<untitled> * | [main.py] | led_bultin.py | test3.py | led2_pwm.py * | [led.py]

```python
# Construct PWM object, with LED on Pin(2).
pwm1 = PWM(Pin(2))
# Set the PWM frequency.
pwm1.freq(1000)
# Set the Duty Cycle Value between 0 - 65535 i.e. 16 bit value.

while True:
    for i in range(0,65535,50):
        pwm1.duty_u16(i)
        time.sleep(0.001)

    print("100% ON")
    time.sleep(2)

    for i in range(65535, 0, -50):
        pwm1.duty_u16(i)
        time.sleep(0.001)

    print("100% OFF")
    time.sleep(2)
```

Shell

```
>>> %Run -c $EDITOR_CONTENT
>>>
```

MicroPython (Raspberry Pi Pico) • COM217

Pulse Width Modulation (PWM) is a technique by which the width of a pulse is varied while keeping the frequency of the wave constant.

PWM Generation

Thonny - C:\Users\Manoj_2020\Documents\led2_pwm.py @ 9 : 1

File Edit View Run Tools Help

<untitled> * | [main.py] | led_bultin.py | test3.py | led2_pwm.py | [led.py]

```
# Construct PWM object, with LED on Pin(2).
pwm1 = PWM(Pin(2))
# Set the PWM frequency.
pwm1.freq(1000)
# Set the Duty Cycle Value between 0 - 65535 i.e. 16 bit value.

while True:
    for i in range(0,65535,50):
        pwm1.duty_u16(i)
        time.sleep(0.001)

    print("100% ON")
    time.sleep(2)

    for i in range(65535, 0, -50):
        pwm1.duty_u16(i)
        time.sleep(0.001)

    print("100% OFF")
    time.sleep(2)
```

Shell

```
100% ON
100% OFF
100% ON
```

MicroPython (Raspberry Pi Pico) • COM217

Pulse Width Modulation (PWM) is a technique by which the width of a pulse is varied while keeping the frequency of the wave constant.

PWM Generation