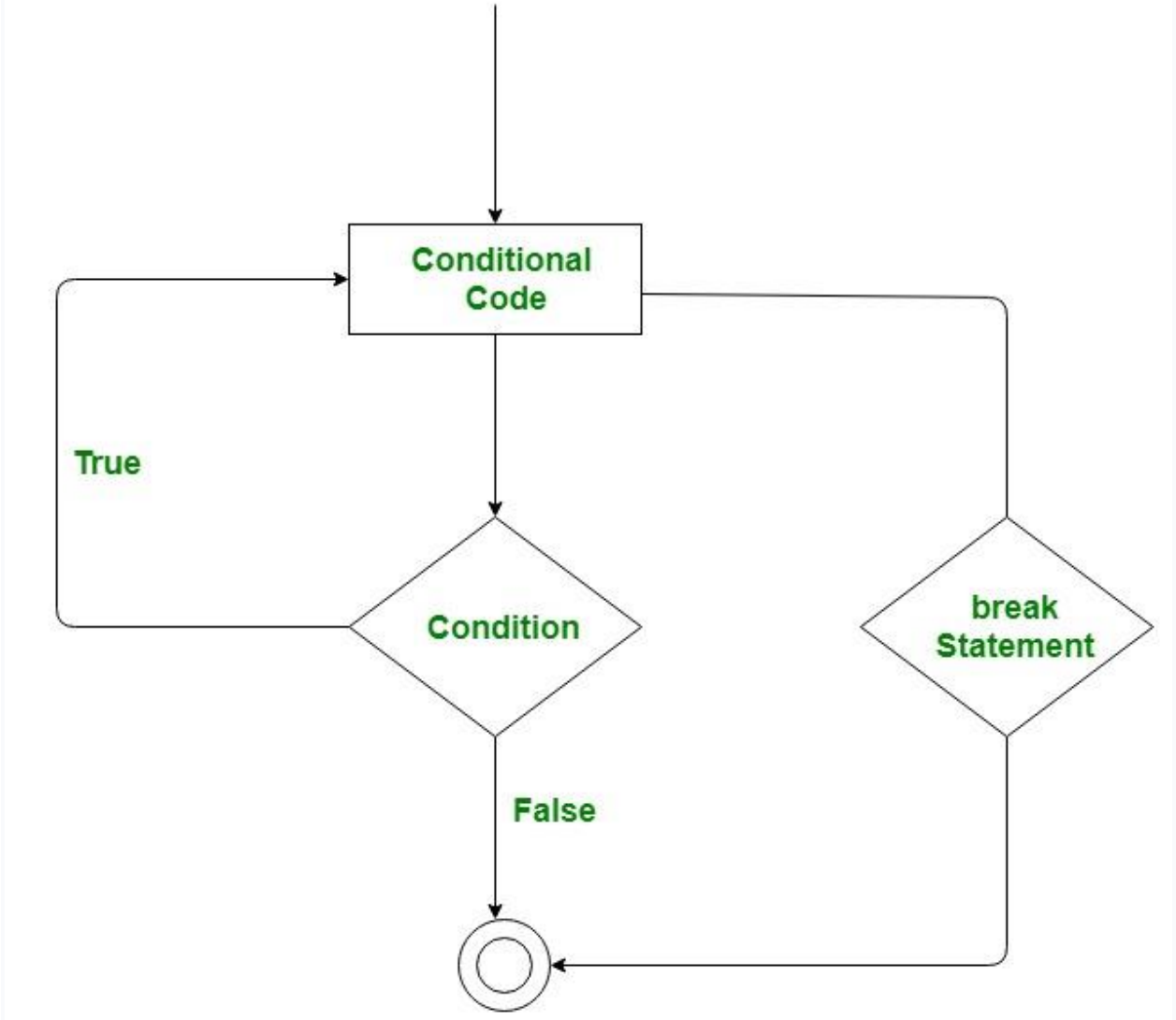
Practical Session with Python:

Use of Python break and continue

In programming, the `break` and `continue` statements are used to alter the flow of loops:

- **`break`** exits the loop entirely
- **`continue`** skips the current iteration and proceeds to the next one



```
for val in sequence:
    # code
    if condition:
        break

    # code
```

- Example 1: For loop and break statement

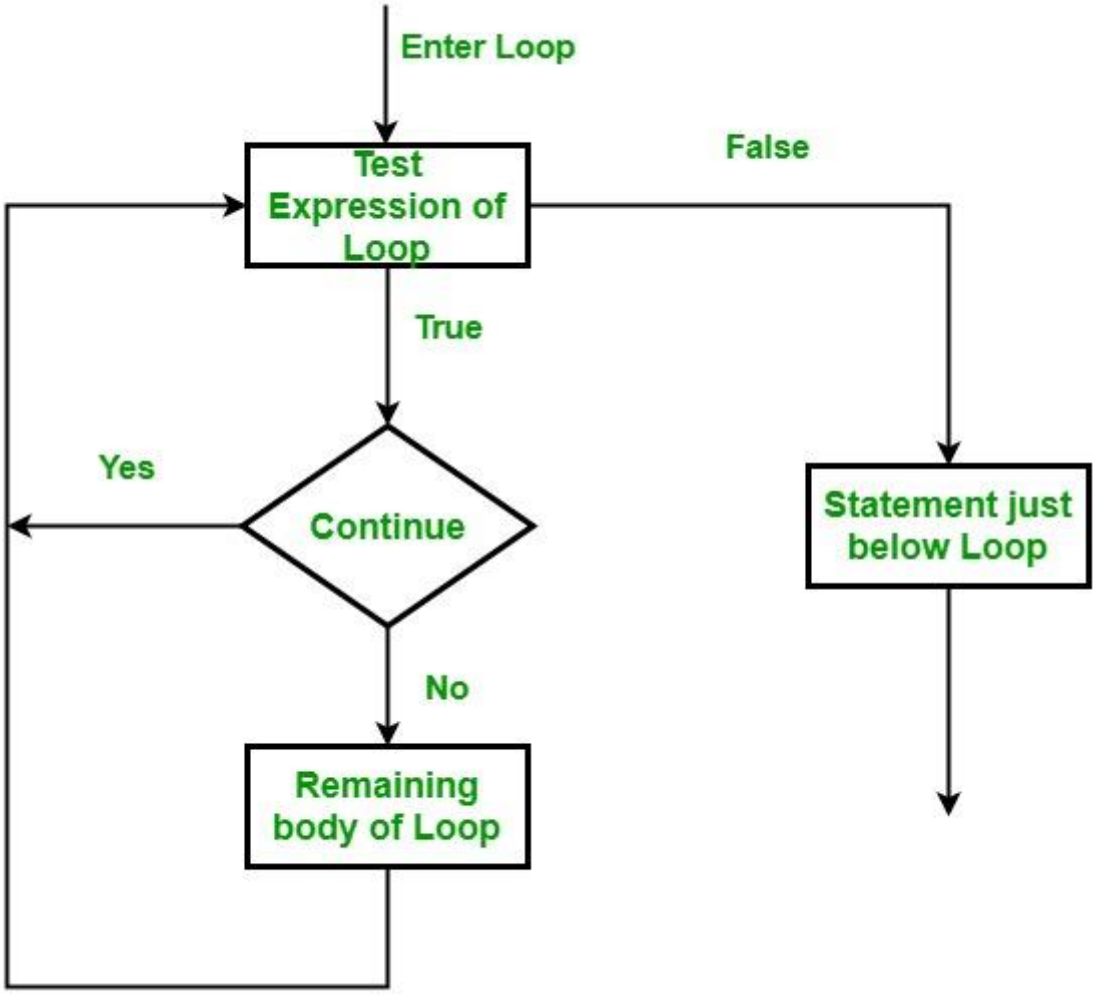
```
for x in range(9):
    if x == 4:
        break
    print(x)
```

```
while condition:
    # code
    if condition:
        break

    # code
```

Example 2: while loop and break statement

```
print("While loop and break")
i = 0
while i < 9:
    if i == 3:
        break
    print(i)
    i += 1
```




```
Example 3:while loop and continue statement
print("Print old number from 1 to 10")
i = 0
while i < 10:
    i += 1
    if i%2 == 0:
        continue
    print(i)
```

Example 4:while loop and continue statement

```
print("Print even number from 1 to 10")
n = int(input("Enter a number"))
i = -1
while i < n:
    i += 1
    if i%2 == 1:
        continue
    print(i)
```

What is PIP?

PIP is a package manager for Python packages, or modules

PIP->Performance Improvement Plan or Package Installer for Python

PIP->

What is a Package?

A package contains all the files you need for a module.

Modules are Python code libraries you can include in your project.

Check if PIP is Installed

pip --version #pip (python 3.11)

pip –V

For Windows:

python get-pip.py

In Linux: sudo apt-get install python-pip

Upgrade Command:

python –m pip install –upgrade pip

#-m option if installed then uninstall old version. After that install new version of pip

py –m pip install –upgrade pip

pip install PackageName # camelcase

pip show camelcase #it show all metadata regarding package camelcase

pip install PackageName==Specific Version

pip list --outdated --verbose

pip help

https://www.knowledgehut.com/blog/programming/what-is-pip-in-python

Source of python repository (PyPi and Github)

https://pypi.org/

Example 5:

```
import camelcase
c = camelcase.CamelCase()
txt = "welcome to edge project kuet"
print(c.hump(txt))
```

Remove a Package

Use the `uninstall` command to remove a package:

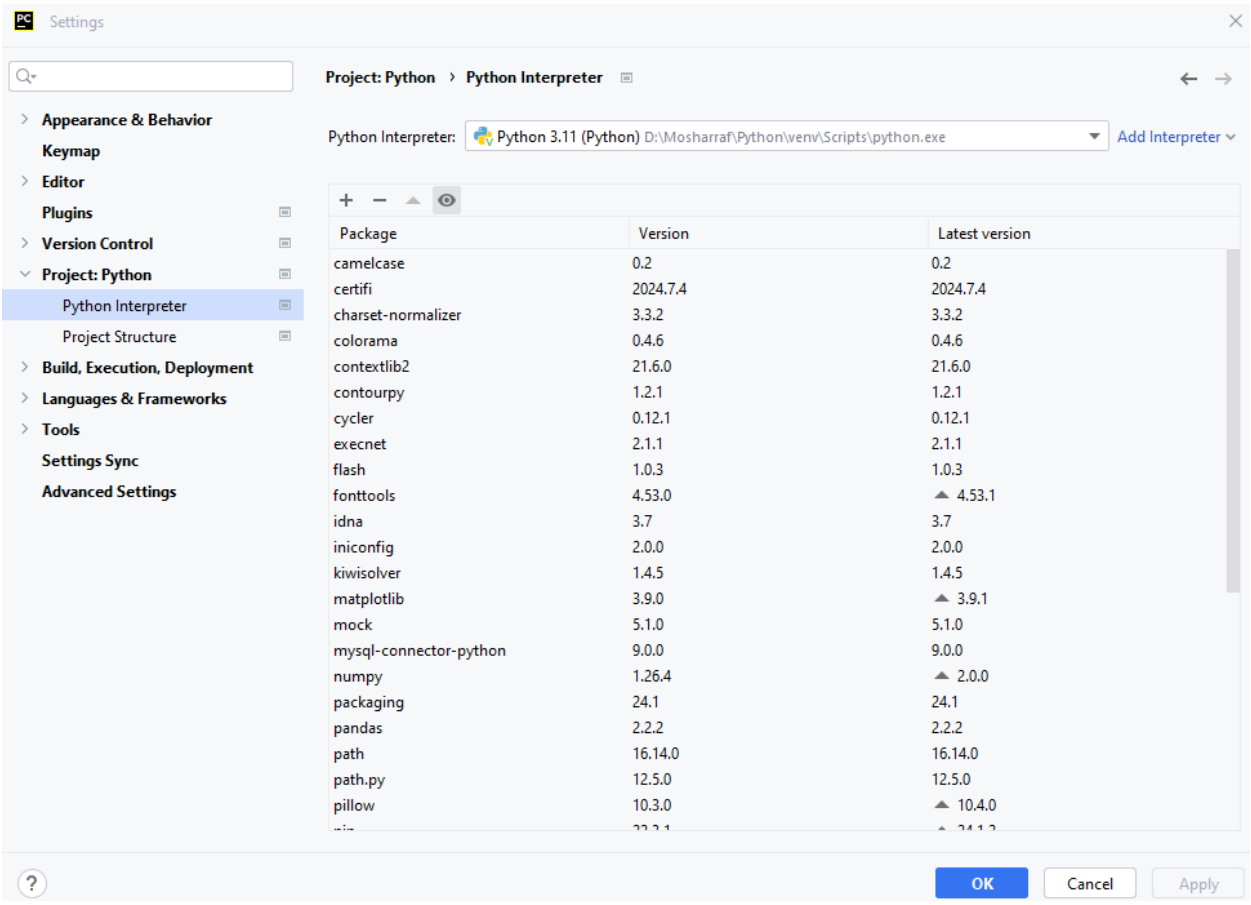
pip uninstall camelcase

List Packages

Use the `list` command to list all the packages installed on your system:

```
pip list
Package                 Version
----------------------- -----------
camelcase                 0.2
cycler                  0.12.1
flash                   1.0.3
matplotlib                3.9.0
```

Pycharm->File ->option->Project: Python->Python Interpreter



Python RegEx

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

RegEx Module

Python has a built-in package called `re`, which can be used to work with Regular Expressions.

Import the `re` module:

RegEx Functions

The `re` module offers a set of functions that allows us to search a string for a match:

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

Example 6:

```
import re
txt = "The rain in Spain and Ukraine"
x = re.findall("ai", txt)
print(x)
```

Example 7a:

```
txt = "The Strom in Spain"
x = re.search("Spain", txt)
print(x)
```

Example 7b:

```
txt = "The Strom in Spain"
x = re.search("Canada", txt)
print(x)
```

Example 7c:

```
txt = "hossain55@gmail.com"
x = re.split("\@", txt)
print(x)

x = re.sub("\s", "   ", txt)
print(x)
```

Metacharacters

Metacharacters are characters with a special meaning:

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"

.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"planet$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"
{}	Exactly the specified number of occurrences	"he.{2}o"
|	Either or	"falls|stays"
()	Capture and group	

Sets

A set is a set of characters inside a pair of square brackets `[]` with a special meaning:

Set	Description
[arn]	Returns a match where one of the specified characters (`a`, `r`, or `n`) is present
[a-n]	Returns a match for any lower case character, alphabetically between `a` and `n`
[^arn]	Returns a match for any character EXCEPT `a`, `r`, and `n`
[0123]	Returns a match where any of the specified digits (`0`, `1`, `2`, or `3`) are present
[0-9]	Returns a match for any digit between `0` and `9`
[0-5][0-9]	Returns a match for any two-digit numbers from `00` and `59`

[a-zA-Z]	Returns a match for any character alphabetically between a and z, lower case OR upper case
[+]	In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string

Example 8: Write a program using a function and regular expression which validate email address. Separate user and domain name

```
import re
def validate_email(email):
    if re.match(f"[^@]+@[^@]+\.[^@]+", email):
        return True
    return False

email = input("Enter your email address: ")
loc = email.find("@")
username = email[:loc]
domain = email[loc+1:]
print("User name is:", username)
print("Domain name is:", domain)

if validate_email(email):
    print(email, "is a valid address")
else:
    print(email, "is not a valid address")
```

Description:

The regular expression `[^@]+@[^@]+\.[^@]+` is commonly used to match email addresses. Here's a breakdown of the syntax:

- `[^@]+`: This matches one or more characters that are not the '@' symbol. `[^@]` is a negated character class that matches any character except '@', and the `+` quantifier means one or more of these characters.
- `@`: This matches the '@' symbol itself.
- `[^@]+`: Again, this matches one or more characters that are not the '@' symbol, representing the domain name part before the period.
- `\.`: This matches the literal '.' character. In regular expressions, the dot is a special character that matches any character, so it needs to be escaped with a backslash (`\.`) to match a literal dot.
- `[^@]+`: This matches one or more characters that are not the '@' symbol, representing the top-level domain (TLD) part after the period.

Exampe 9:

Write a python program that takes multiple lines as input and save it to a filename “output.txt”. When press “Done” in newline program exit.

```
def write_lines_to_file(filename):
    lines = []
    print("Enter your lines of text (type 'DONE' on a new line to finish):")
    while True:
        line = input()
        if line.strip().upper() =='DONE':
            break
        lines.append(line)
```

```
    with open(filename,"w") as file:
        for line in lines:
            file.write(line + "\n")
    print(f"Text Successfully written to
file named: {filename}")

filename = "output1.txt"  # Specify the
file name
write_lines_to_file(filename)
# See write_lines_to_file

Example 10a: Write a program to search a
desired text in the file of example 9
created
import re
# Specify the file name and the regular
expression pattern
file_name = 'output.txt'
pattern = r'line\s+of'

# Open the file and search for the pattern
with open(file_name, 'r') as file:
    lines = file.readlines()

# Check each line for the pattern
for line_number, line in enumerate(lines,
start=1):
    if re.search(pattern, line):
        print(f"Found pattern '{pattern}'
in line {line_number}: {line.strip()}")
```

Example 10b: Write a program to search a desired text in the file of example 9 created

```
file_name = 'output.txt'
search_term = 'line'

with open(file_name, 'r') as file:
    lines = file.readlines()

   # Check each line for the search term
for line_number, line in enumerate(lines,
start=1):
    if search_term in line:
        print(f"Found '{search_term}' in line
{line_number}: {line.strip()}")
```

Modules

A module is a single Python file containing a set of functions, classes, or variables. Modules allow you to logically organize your Python code.

Creating a Module

Create a file named `myModule.py`:

Example 11:

```
def greeting(name):
  print("Hello: " + name)


def add(a, b):
  return a+b

person1 = {
```

```
  "Name": "Mosarraf",
  "Age": 40,
  "District": "Satkhira"
}
```

Use the module in another Python file:

```
import myModule
print(myModule.person1["Age"])
print(myModule.add(6,7)
```

Using from and import

```
from myModule import greeting, add
greeting("Mosarraf")
print(add(10, 20))    # Output: 30
```

Import only the person1 dictionary from the module:

```
from mymodule import person1
print (person1["age"])
```

Packages

A package is a collection of Python modules. Packages allow you to structure your project in a hierarchical manner.

Creating a Package

1. Create a directory named `mypackage`.
2. Inside `mypackage`, create an empty file named `__init__.py` to indicate that this directory is a package. The `__init__.py` file can also execute initialization code for the package.
3. Create a module inside the package, e.g., `module1.py`:

1. Create a module inside the package, e.g., `module1.py`:

```
# myPackage/myModule1.py
2. def fnSwapCase():
    name = "My Name is Engr. Md. Mosarraf
Hossain Sarder"
    x = name.swapcase()
    print(x)
```

3. Create another module, e.g., my`Module2.py`:

```
mypackage/module2.py
def fnReverseNumber():
    n = int(input("Enter a number: "))
    l = len(str(n))
    # print(l)

    for i in range(l):
        x = n % 10
        y = int(n / 10)
        print(x, end='')
        n = y
```

4. Create another module e.g, myModule3.py

```
def fibo(n):
    a, b = 0, 1
    print(a)
    x = 0

    while x <= n:
        c = a + b
        x = x + 1
        print(c)
        a = b
        b = c
```

Using the Package

Import the package modules in another Python file:

Example 12:

```
from myPackage import myModule1, myModule2,
myModule3
myModule1.fnSwapCase()
myModule2.fnReverseNumber()
myModule3.fibo()
```

Summary

- **Module**: A single file containing Python code (functions, classes, variables).
- **Package**: A collection of modules organized in directories, potentially with subpackages.

Naming a Module

You can name the module file whatever you like, but it must have the file extension `.py`

Re-naming a Module

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

Built-in Modules

Example 13:

```
import platform

x = platform.system()
print(x)
```

Using the dir() Function

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

Example 14:

```
import platform

x = dir(platform)
print(x)
```

A **script** is a Python file that's intended to be run directly. When you run it, it should do *something*. This means that scripts will often contain code written outside the scope of any classes or functions.

A **module** is a Python file that's intended to be imported into scripts or other modules. It often defines members like classes, functions, and variables intended to be used in other files that import it.

A **package** is a collection of related modules that work together to provide certain functionality. These modules are contained within a folder and can be imported just like any other modules. This folder will often contain a special `__init__` file that tells Python it's a package, potentially containing more modules nested within subfolders

A **library** is an umbrella term that loosely means "a bundle of code." These can have tens or even hundreds of individual modules that can provide a wide range of functionality. Matplotlib is a plotting library. The Python Standard Library contains hundreds of modules for performing common tasks, like sending emails or reading JSON data. What's special about the Standard Library is that it comes bundled with your installation of Python, so you can use its modules without having to download them from anywhere.

Source: https://realpython.com/lessons/scripts-modules-packages-and-libraries/

Script

A script is a file containing a sequence of Python commands that are meant to be executed directly. It's typically written for a specific task and can be run as a standalone program.

Example:

```
# hello.py
def greet(name):
    return f"Hello, {name}!"

if __name__ == "__main__":
    name = input("Enter your name: ")
    print(greet(name))
```

To run this script, you would execute `python hello.py` from the command line. It prompts the user for their name and then prints a greeting.

Module

A module is a file containing Python definitions and statements intended for use in other Python programs. A module can define functions, classes, and variables that can be imported and used in other modules or scripts.

Examples of modules:

1. Datetime
2. Regex
3. Random etc.

Example:

```
# my_module.py
def add(a, b):
    return a + b
```

```
def subtract(a, b):
    return a - b
```

You can import and use this module in another script:

```
# use_module.py
import my_module

print(my_module.add(5, 3))  # Output: 8
print(my_module.subtract(5, 3))  # Output: 2
```

Package

A package is a way of structuring Python's module namespace by using "dotted module names". A package is a directory that contains a special `__init__.py` file and one or more module files.

Examples of Packages:

1. Numpy
2. Pandas
3. SciPy

Example:

```
my_package/
    __init__.py
    module1.py
    module2.py
```

`__init__.py` can be an empty file or can contain initialization code for the package.

module1.py:

```
# module1.py
def foo():
    return "foo from module1"
```

module2.py:

```
# module2.py
def bar():
    return "bar from module2"
```

You can import and use this package in another script:

```
# use_package.py
from my_package import module1, module2

print(module1.foo())  # Output: foo from module1
print(module2.bar())  # Output: bar from module2
```

Library

A library is a collection of modules and packages that are bundled together for a particular purpose. Libraries provide a range of functions and classes that can be used to develop applications. Libraries are often distributed via package managers like `pip`.

Examples of Libraries:

1. Matplotlib
2. Pytorch
3. Pygame
4. Seaborn etc.

Example:

NumPy is a popular library for numerical computations.

To install NumPy:

```
pip install numpy
```

To use NumPy in your script:

```
import numpy as np

array = np.array([1, 2, 3, 4, 5])
print(array)  # Output: [1 2 3 4 5]
print(np.mean(array))  # Output: 3.0
```

Summary

- **Script:** A file with Python code meant to be executed directly.
- **Module:** A file with Python code meant to be imported and used in other Python code.
- **Package:** A directory with multiple modules and an `__init__.py` file, used for organizing related modules.
- **Library:** A collection of modules and packages providing a wide range of functionalities, often for a specific purpose.