

Lecture 04

Conditional Structure

What is a conditional structure?

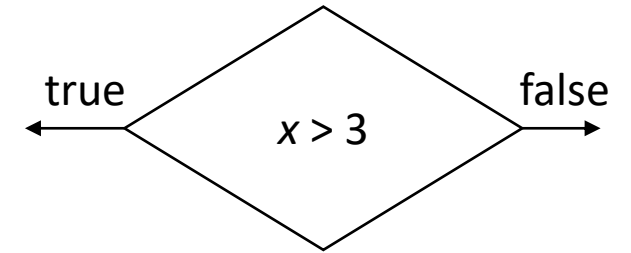
- A structure, defined in a programming language, that contains a conditional clause and its associated statements of the consequence(s).
 - A decision based on the given condition will be made to determine the consequence.
- In Python, it is referred to as the **if** statement.
 - Variations (*next few slides*)
- What about the **switch..case** statement?
 - Python 3.10 introduces a **match..case** statements
 - Yet, programmers can simulate the **switch..case** structure using a Python dictionary

Basic *if..else* Structure

- The following illustrates a generic if structure

```
if (condition)  
  codes to run when condition is true  
else  
  code to run when condition is false
```

- where,
 - *condition* must be Boolean expression



Basic Syntax of *if* statement

- Without ***else*** clause:

```
if (condition):  
    statement(s) for true
```

- *condition* must be a Boolean expression. E.g. ($x < 5$)

- With ***else*** clause:

```
if (condition):  
    statement(s) for true  
else:  
    statement(s) for false
```

```
if weight > 50:  
    print("A $25 charge applies.")
```

```
temperature = float(input("What is the temperature? "))  
  
if temperature > 70:  
    print('Wear shorts.')  
else:  
    print('Wear long pants.')
```

The *elif* clause

- A Python way to combine else and if for saying "if the previous conditions were not true, then try this condition".
- You can have as many *elif* clauses in an *if* structure as needed.

```
n1 = float(input("Enter the 1st number: "))
n2 = float(input("Enter the 2nd number: "))

if (n1 > n2) :
    print(n1, "is larger.")
elif (n1 < n2):
    print(n2, "is larger.")
else:
    print("They are equal.")
```

elif ≠ *else if*

Python does not support "else if"

Nested *if* in Python

- The term “nested” means “one in another”.
- There may be a situation when you want to check for another condition after a condition resolves to true.
 - nested *if* could be a solution
- Nesting can happen within the *if* or *else* clauses.

```
if (condition1):  
    if (condition2):  
        execution1  
    else:  
        execution2  
else:  
    executuin3
```

```
if (condition1):  
    execution1  
else:  
    if (condition2):  
        executuin2  
    else:  
        execution3
```

Score	Grade
90 ~	A
80 ~ 89.99	B
70 ~ 79.99	C
60 ~ 69.99	D
~ 60	F

Sample codes

- Nesting can happen within the *if* or *else* clauses.

Score	Grade
90 ~	A
80 ~ 89.99	B
70 ~ 79.99	C
60 ~ 69.99	D
~ 60	F

```
gd = float(input("Enter a  
score: "))
```

```
if (gd < 90):  
    if (gd < 80):  
        if (gd < 70):  
            if (gd < 60):  
                print("F")  
            else:  
                print("D")  
        else:  
            print("C")  
    else:  
        print("B")  
else:  
    print("A")
```

```
gd = float(input("Enter  
a score: "))
```

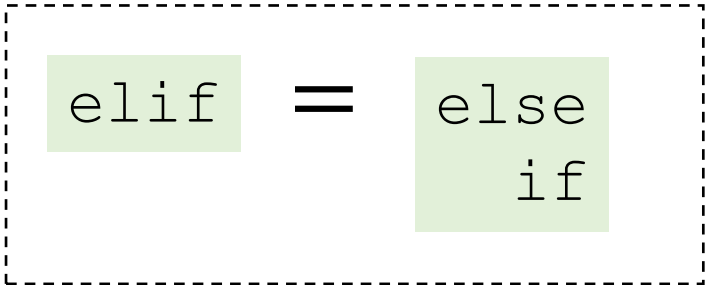
```
if (gd >= 90):  
    print("A")  
else:  
    if (gd >= 80):  
        print("B")  
    else:  
        if (gd >= 70):  
            print("C")  
        else:  
            if (gd >= 60):  
                print("D")  
            else:  
                print("F")
```

The *elif* clause

- A Python way to combine *else* with a lower-layer *if* for saying “if the previous conditions were not true, then try this condition”.
- You can have as many *elif* clauses in an if structure as needed.

```
n1 = float(input("Enter the 1st number: "))
n2 = float(input("Enter the 2nd number: "))

if (n1 > n2):
    print(n1, "is larger.")
elif (n1 < n2):
    print(n2, "is larger.")
else:
    print("They are equal.")
```



A diagram enclosed in a dashed rectangular border. It shows the word `elif` in a light green box, followed by an equals sign `=`, followed by the words `else` and `if` stacked vertically in another light green box.

Boolean expressions could be hybrid

- In Python, the Boolean expression of an if structure could be a hybrid expression.
 - “Hybrid” implies that two or more Boolean expressions are combined to form a single one

```
x = int(input("Enter an integer between 1 and 47: "))

if ((x < 2) or (x > 46)):
    print("Invalid number.")
else:
    print("Valid number.")
```

In-Class Exercise

- Write a program to calculate the total amount to pay after discount.

Cost	Discount
~ 25.99	0
26 ~ 44.99	5%
45 ~ 69.99	15%
70 ~	20%

```
tc = float(input("Enter the
total: "))

if (tc >= 70):
    tc = tc*(1-0.2)
else:
    if (tc >= 45):
        tc = tc*(1-0.15)
    else:
        if (tc >= 26):
            tc = tc*(1-0.05)
        else:
            tc = tc

print(tc)
```

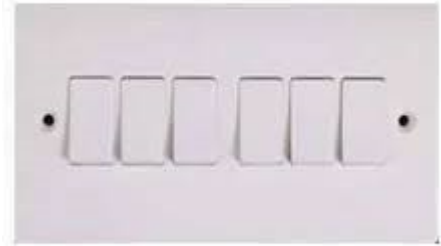
match..case

The Python's approach for traditional switch..case statement

The *switch..case* Structure

- The following is generic *switch..case* structure used by programming languages like Java, C++, and C#.

```
switch (expression)
{
    case 0: execution0; break;
    case 1: execution1; break;
    .....
    case n: executionn; break;
    default:
}
```



Sample Codes (C++ and Java)

```
public class Sample
{
    public static void main(String[] args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter blood type: ");
        String s = sc.nextLine();
        char ch = s.charAt(0);
        switch (ch)
        {
            case 'A': System.out.println("Eat apples."); break;
            case 'B': System.out.println("Eat banana."); break;
            case 'O': System.out.println("Eat orange."); break;
            case 'AB': System.out.println("Eat apple and banana."); break;
            default: System.out.println("No such blood type.");
        }
    }
}
```

```
#include <iostream>
using namespace std;

int main()
{
    int bt=0;
    cout << "Enter your blood type [0-A, 1-B, 2-O, 3-AB]: ";
    cin >> bt;

    switch (bt)
    {
        case 0: cout << "Eat apples." << endl; break;
        case 1: cout << "Eat banana." << endl; break;
        case 2: cout << "Eat orange." << endl; break;
        case 3: cout << "Eat apple and banana." << endl; break;
        default: cout << "No such blood type." << endl;
    }

    return 0;
}
```

The *match..case* statement in Python

- Python 3.10 introduced the *match..case* structure with the following syntax:

```
match (object):  
    case expression0:  
        statements  
    case expression1:  
        statements  
    .....  
    case _:  
        statements
```

where,

- *object*: A code object that produces a value for the “match” keyword to find a matched case.
- *Expression[0-n]*: A unique value or a unique collection of values.
- *_*: A special case to be used when there is no matched case. Optional


- Python’s *match..case* goes beyond traditional *switch..case* statement
 - It includes many Python-specific features

match..case > swtich..case

The new "*match..case*" structure

- Python 3.10 introduced a new structure -- "match..case" which could be functionally similar to the traditional "switch..case".
- Interestingly, it does not require the "break" statement to end a case (next slide)
- Syntax:

```
match variable:  
    case label1:  
    case label2:  
    .....  
    _:
```



It uses an underscore (_) to denote the "default" case.

```
bt = input("Enter your blood type: ")  
  
match bt:  
    case "A": print("Eat apples.")  
    case "B": print("Eat banana.")  
    case "O": print("Eat orange.")  
    case "AB": print("Eat apple and banana.")  
    case _: print("No such blood type.")
```

What can the “object” be in *match..case*?

- A variable that represents
 - A scalar value (int, float, bool, string, etc.)
 - A sequence structure (Python list or tuple)
- An expression (e.g. `n%2`)
- and more

```
ft = input("Enter the fruit name: ")

match ft: #sequence of characters
    case 'apple': print("APPLE")
    case 'orange': print("ORANGE.")
    case _: print("No defined.")
```

```
match (3>5):
    case True: print("Correct.")
    case False: print("Incorrect.")
```

```
n = int(input("Enter an integer: "))

match n%2: #expression
    case 0: print("Even number.")
    case 1: print("Odd Number.")
```

```
s = input("Enter three colors: ")
s = s.split()

match s: #a Python list
    case ["red", "green", "blue"]:
        print("Group 1")
    case ["white", "orange", "purple"]:
        print("Group 2")
```

```
x = 4
match (2*x-16):
    case i if (i>=0): print("Positive.")
    case i if (i<0): print("Negative.")
```


When to use the “_” case

- The “_” case specifies what to do to run if there is no matched case.
 - Use it when unexpected value may be given to the *match..case* statement.

```
from datetime import datetime
dt = datetime.now()
wd = dt.weekday()
```

```
s = "Today is "
```

```
match wd:
    case 0: s += "Monday"
    case 1: s += "Tuesday"
    case 2: s += "Wednesday"
    case 3: s += "Thursday"
    case 4: s += "Friday"
    case 5: s += "Saturday"
    case 6: s += "Sunday"
```

```
print(s)
```

```
wd = int(input("Enter a integer [0-6]: "))
s = "Today is "
```

```
match wd:
    case 0: s += "Monday"
    case 1: s += "Tuesday"
    case 2: s += "Wednesday"
    case 3: s += "Thursday"
    case 4: s += "Friday"
    case 5: s += "Saturday"
    case 6: s += "Sunday"
    case _: s = "Invalid number."
```

```
print(s)
```

```
Enter a integer [0-6]: 9
Invalid number.
```

```
>>>
```

“match..case” sample 01

In this example,

- “bt” is a variable
- Every case is defined by a string.

```
bt = input("Enter your blood type: ")  
bt = bt.upper()
```



```
match bt:
```

```
    case "A": print("Eat more apples.")  
    case "B": print("Eat more bananas.")  
    case "O": print("Eat more oranges.")  
    case "AB": print("Eat more apples and bananas.")  
    case _: print("No such blood type.")
```

```
>>>   
=====
Enter your blood type: a
Eat more apples.
>>>   
=====
Enter your blood type: b
Eat more bananas.
>>>   
=====
Enter your blood type: o
Eat more oranges.
>>>   
=====
Enter your blood type: ab
Eat more apples and bananas.
>>>   
=====
Enter your blood type: c
No such blood type.
>>> |
```

“match..case” sample 02

- In this example,
 - The pipe (“|”) denotes the “or” operator
 - cid[3:6] extracts out the last 3 characters

main.py	 	Save	Run	Shell
<pre>1 cid = input("Enter the course ID: ") 2 cid = int(cid[3:6]) #extract the digits 3 4 match cid: 5 case 245 246 247: print("Programming") 6 case 110 243: print("Operation System") 7 case _: print("Not available.") 8</pre>				<pre>Enter the course ID: cis243 Operation System > </pre>

```
cid = input("Enter the course ID: ")
cid = int(cid[3:6]) #extract the digits

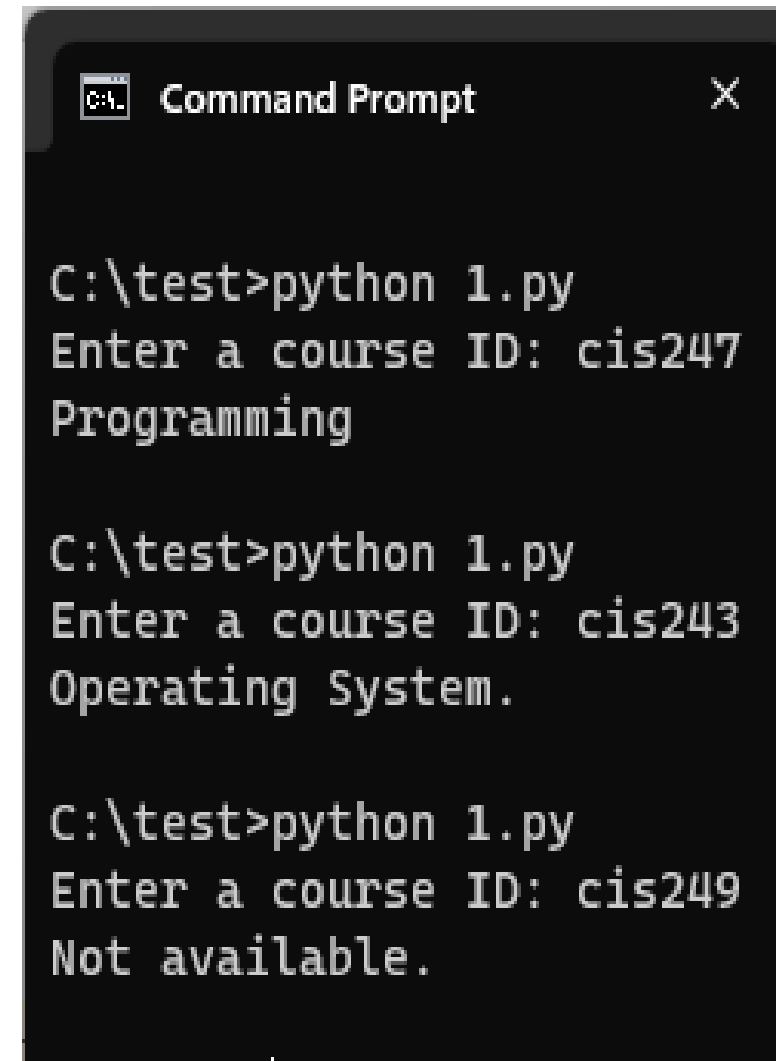
match cid:
    case 245 | 246 | 247: print("Programming")
    case 110 | 243: print("Operation System")
    case _: print("Not available.")
```

“*match..case*” sample 03

- In this example,
 - *i* is a variable defined in two cases for an if statement to check if the value of *i* is in the list or tuple.

```
clr = input("Enter a course ID: ")

match clr.lower():
    case i if i in ("cis245", "cis246", "cis247"):
        print("Programming")
    case i if i in ['cis110', 'cis243']:
        print("Operating System.")
    case _:
        print("Not available.")
```



```
Command Prompt

C:\test>python 1.py
Enter a course ID: cis247
Programming

C:\test>python 1.py
Enter a course ID: cis243
Operating System.

C:\test>python 1.py
Enter a course ID: cis249
Not available.
```

Simulating the traditional *switch..case* with a Python Dictionary

Prior to Python 3.10

An alternative of *switch..case* statement

- Prior to Python 3.10, Python did not support ***switch..case*** statement.
- Use a Python “dictionary” to simulate it.
 - A “dictionary” use a pair of *key* and *value* to define every element.
 - Let the *key* be the “case label”, and let the *value* be the case statement.

```
///traditional
switch(gd)
{
    case 'A': str = "Eat more apples."; break;
    case 'B': str = "Eat more bananas."; break;
    case 'O': str = "Eat more oranges."; break;
    case 'AB': str = "Eat more avocados."; break;
}
```

```
####Python
switch = {'A': "Eat more apples.",
          'B': "Eat more bananas.",
          'O': "Eat more orange.",
          'AB': "Eat more avocados."}

gd = input("Enter your blood type: ")

print(switch[gd])
```

Sample code – *switch..case* simulation 1

- In Python, the “weekday()” function of the “datetime” module only returns an integer, not the name of week day, with 0 indicating Monday, 1 indicating Tuesday, and so on.
- Create a Python dictionary with the following elements to simulate a “*switch..case*” structure.
 - 0 is the “case label” of the case statement “Monday”.

```
switch = {  
    0 : "Monday",  
    1 : "Tuesday",  
    2 : "Wednesday",  
    3 : "Thursday",  
    4 : "Friday",  
    5 : "Saturday",  
    6 : "Sunday" }
```

```
import datetime  
now = datetime.datetime.now()  
  
wd = now.weekday()  
  
switch = { 0 : "Monday", 1 : "Tuesday",  
          2 : "Wednesday", 3 : "Thursday",  
          4 : "Friday", 5 : "Saturday",  
          6 : "Sunday" }  
  
print("Today is", switch[wd])
```

Sample code – *switch..case* simulation 2

- Write a Python program that will use a Python dictionary to simulate a *switch..case* structure to randomly display “tip of the day” based on the following table.

<i>n</i>	Tip
0	Failure is the path of lease persistence.
1	Now is the time to try something new.
2	Success is failure turned inside out.
3	You have yearning for perfection.
4	Those who care will make the effort.
5	Practice makes perfect.
6	Good news will come to you by mail.

```
import random

n = random.randint(0, 6)

switch = {
    0: "Failure is the path of lease persistence.",
    1: "Now is the time to try something new.",
    2: "Success is failure turned inside out.",
    3: "You have yearning for perfection.",
    4: "Those who care will make the effort.",
    5: "Practice makes perfect.",
    6: "Good news will come to you by mail." }

print("Tip of the day:", switch[n])
```


Sample code – *switch..case* simulation 3

- How to let the “case statement” perform a task?
 - In the previous example, the “case statement” simply return a text.
- Solution – let the value of the Python dictionary be an “function call” that can call a function.
 - A function in Python is a block of “Python code that performs the task” with a unique identifier.

```
def Add() : return (float(n1) + float(n2))
def Sub() : return (float(n1) - float(n2))
def Mul() : return (float(n1) * float(n2))
def Div() : return (float(n1) / float(n2))
def Mod() : return (float(n1) % float(n2))

switch = {
    "+": Add(),
    "-": Sub(),
    "*": Mul(),
    "/": Div(),
    "%": Mod(),
}
```

Combining *if* and *switch..case* in Python

- Scenario:
 - When the case label needs to be a range of values.
- E.g. A company offers discount of a product by quantity as shown below. The MSRP is \$15.25. Write a Python program that ask the user to enter the quantity and return the selling price after discount.

Quantity	Discount Rate
1 ~ 100	0%
101 ~ 200	5%
201 ~ 300	10%
301 ~	15%

```
qty = int(input("Enter the quantity: "))

switch = { 'A': 0.15, 'B': 0.1, 'C': 0.05 }

if (qty >= 301):
    print(format(switch['A']*15.25*qty, ".2f"))
elif (qty <= 201):
    print(format(switch['B']*15.25*qty, ".2f"))
elif (qty <= 101):
    print(format(switch['C']*15.25*qty, ".2f"))
else:
    print(format(15.25*qty, ".2f"))
```

Sample Question

- A special job fair invites applicants with the following conditions:
 - A. Have a GPA ≥ 3.7 , or
 - B. Have at least 90 credit hours
 - C. Or have both A and B

```
gpa = float(input("Enter the GPA: "))
ch = float(input("Enter the total credit hours: "))

if (gpa >= 3.7 or ch >= 90):
    print("Qualified.")
else:
    print("Not qualified.")
```

Sample Boolean Expression

- $(5 > 3)$
- $(x < 6)$
- $(userid == "jlo")$