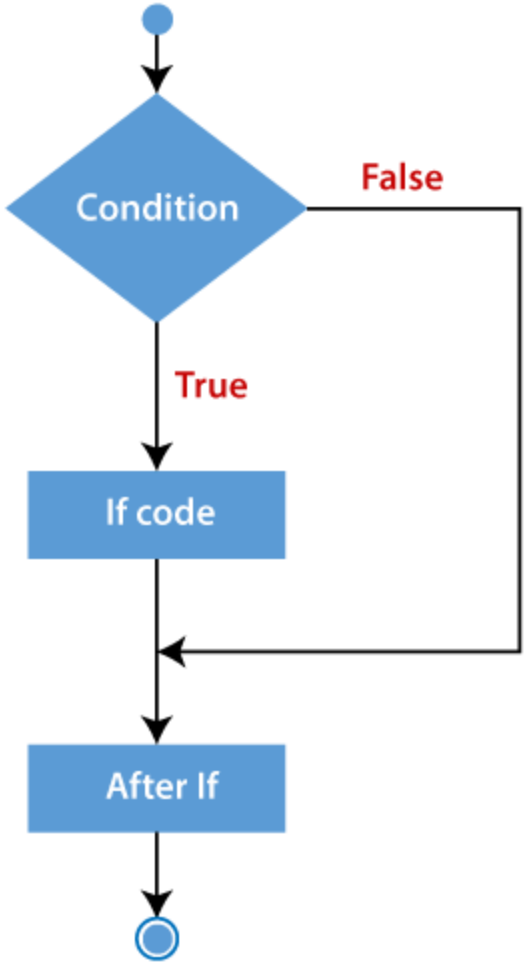
“if” STATEMENTS

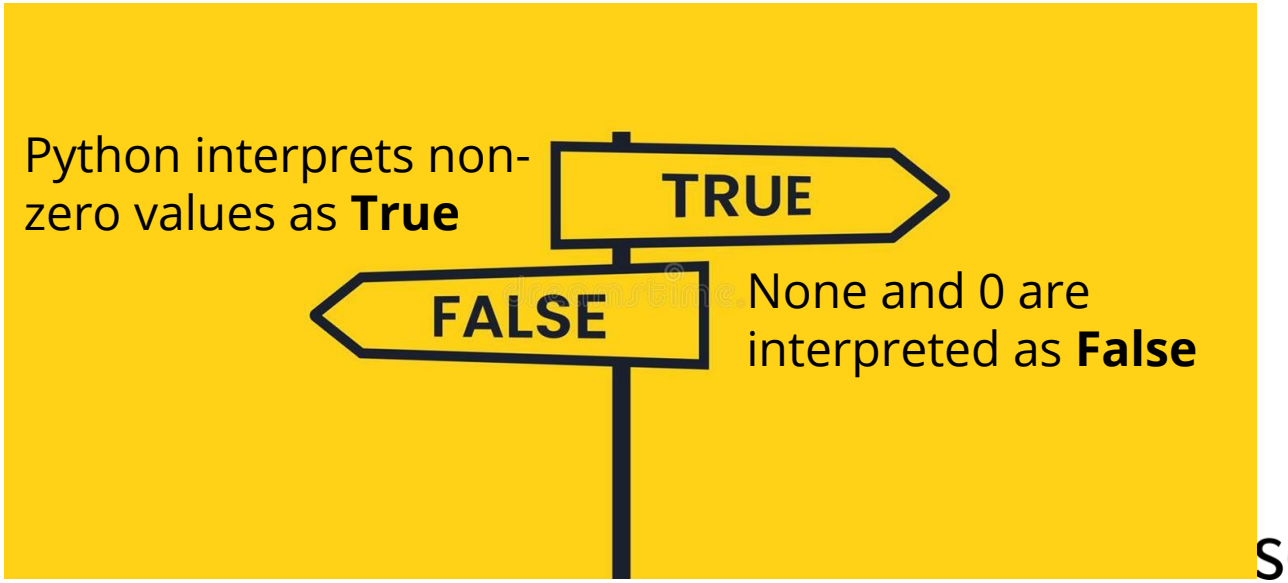
softserve

Python Decision Making



Decision making is required when we want to execute a code only if a certain condition is satisfied.

The **if...elif...else** statement is used in Python for decision making.



softserve

Comparison Operators

Operator	Example a = 5, b = 10
==	(a == b) is False.
!=	(a != b) is True.
>	(a > b) is False.
<	(a < b) is True.
>=	(a >= b) is False.
<=	(a <= b) is True.

softserve

Boolean operators

Logical operators are words **(and, or, not)** not symbols **(&&, ||, !)**.

not	
A	!A
False	True
True	False

and		
A	B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

or		
A	B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

softserve

if statement

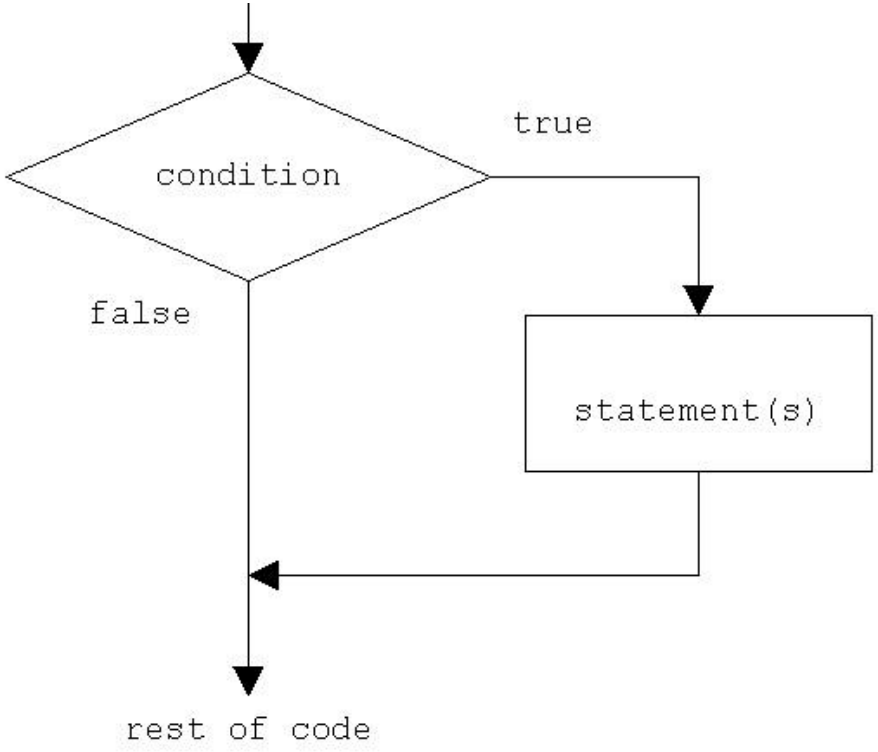
```
if test expression:
   statement(s)
```

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is **True**. If the text expression is **False**, the statement(s) is not executed.

```
score = 12

if score > 8:
    print(“You have passed the exam")

print("Exam was finished.")
```



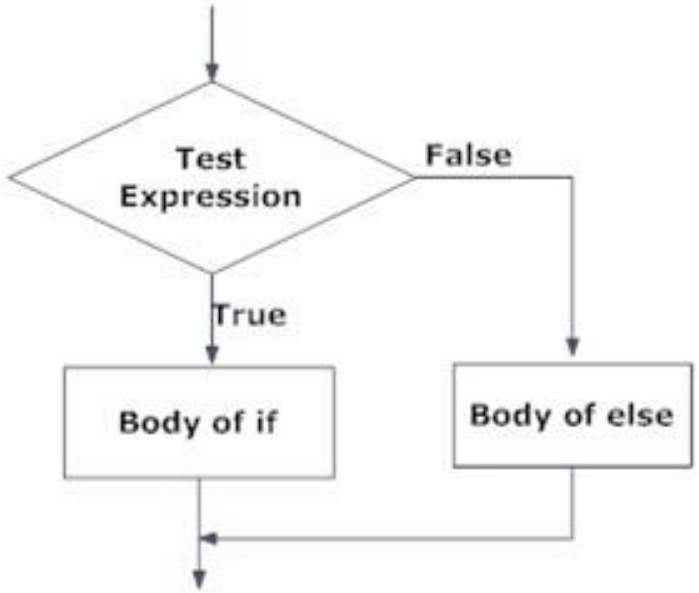
serve

if...else Statement

```
if expression:
   statement(s)
else:
   statement(s)
```

The **if..else** statement evaluates test expression and will execute body of if only when test condition is **True**.

If the condition is **False**, body of else is executed. Indentation is used to separate the blocks.




```
temperature = float(input('What is the temperature? '))

if temperature > 30:
    print('Wear shorts.')
else:
    print('Wear long pants.')
print('Get some exercise outside.')
```

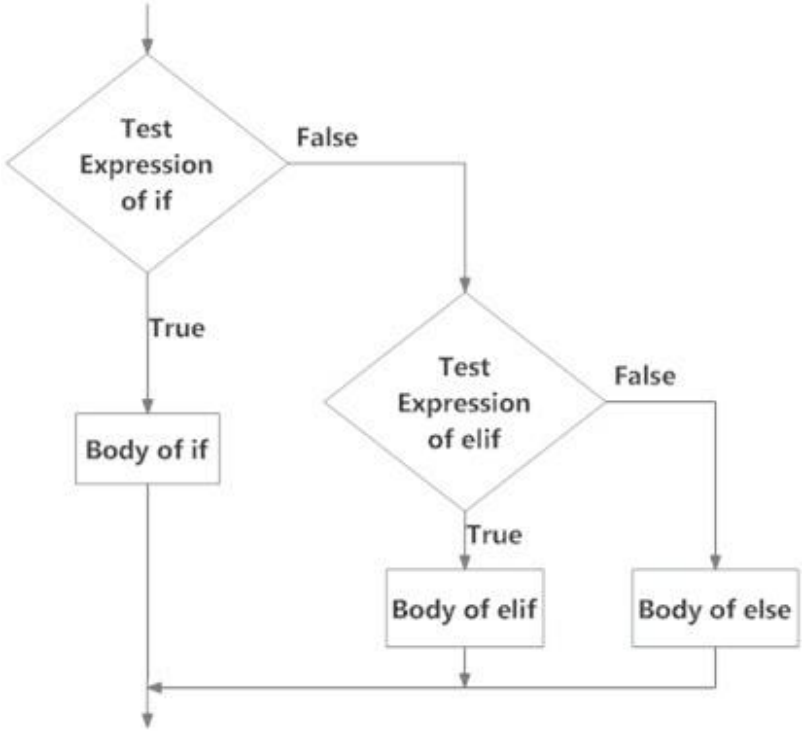
softserve

if...elif...else Statement

```
if expression1:
   statement(s)
elif expression2:
   statement(s)
elif expression3:
   statement(s)
else:
   statement(s
```

The **elif** is short for else if. It allows us to check for multiple expressions.
If the condition for if is **False**, it checks the condition of the next elif block and so on.
If all the conditions are **False**, body of else is executed.
Only one block among the several **if...elif...else** blocks is executed according to the condition.
The if block can have only one else block. But it can have multiple elif blocks.

```
if age < 12:
    print('kid')
elif age < 18:
    print('teenager')
elif age < 50:
    print('adult')
else:
    print('you are not old')
```



softserve

if...elif...else Statement

```
if score >= 90:
 letter = 'A'
else:
# grade must be B, C, D or F
        if score >= 80:
              letter = 'B'
        else: # grade must be C, D or F
                if score >= 70:
                   letter = 'C'
                else: # grade must D or F
                        if score >= 60:
                             letter = 'D'
                        else: letter = 'F‘
```

```
if score >= 90:
        letter = 'A'
elif score >= 80:
        letter = 'B'
elif score >= 70:
        letter = 'C'
elif score >= 60:
        letter = 'D‘
else: letter = 'F'
```

softserve

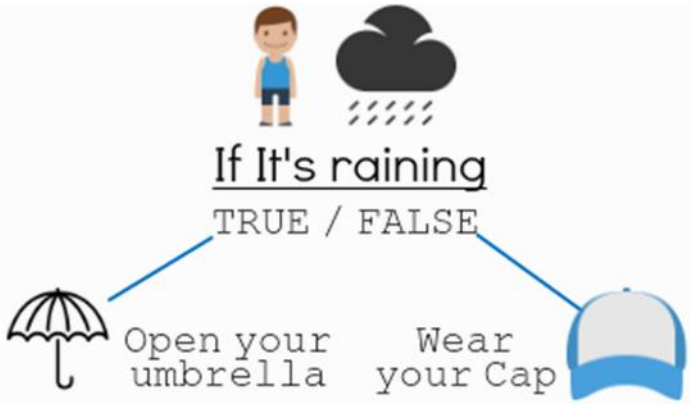
Ternary Operator

statement() **if** condition **else** statement()

```
>>> 'true' if True else 'false'
'true'
>>> 'true' if False else 'false'
'false'
```

```
weather = "raining“

print("Open Your umbrella" if weather == "raining" else "Wear your cap")
```



softserve

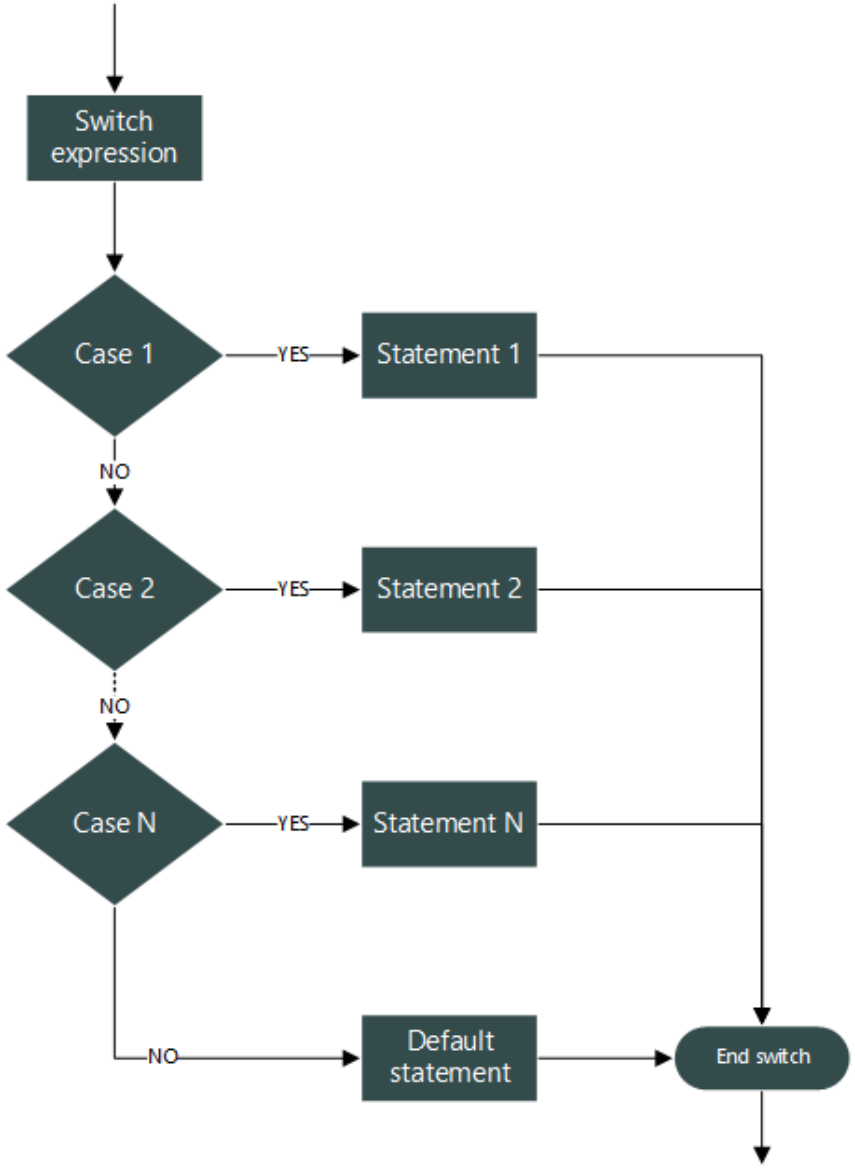
Python does not support construction **switch-case** !

softserve

match..case

“Match case” is analog of “switch case” in Python.

```
match status:
        case 400:
                print "Bad request"
        case 401:
                print "Unauthorized"
        case 403:
                print "Forbidden"
        case 404:
                print "Not found"
        case _:
                print "Other error"
```



softserve

match..case

```
match status:
    case 400:
        print ("Bad request")
    case 401 | 403 as error:
        print (f'{error} is
authentication error')
    case 404:
        print ("Not found")
    case _:
        print ("Other error")
```

In Python, we can use a pipe “|” to combine cases into a single case. It is also considered an “OR” relationship.

Also, we can use “as” keyword followed by a variable, and this variable will be the reference.

To catch default case, we use “_”.

match..case

In “match case” we can have different actions in dependency of different numbers of parameters.

And have “*” to describe unknown number of them.

In some cases, we use variables to take our parameters in further actions.

```
match values:
    case "load", link:
        load(link)
    case "save", link, filename:
        save(link, filename)
    case "save", link,
*filenames:
        for filename in filenames:
            save(link, filename)
    case _:
```

softserve

match..case

In “match cases”, we can put arrays, dictionaries, objects (need __match_args__ attribute) and other parameters.

```
match item:
        case ['evening', action]:
            print(f'You almost finished the day! Now
{action}!')
        case [time, action]:
            print(f'Good {time}! It is time to {action}!')
        case _:
            print('The time is invalid.')
```

array

```
match item:
        case {“time”: 'evening’, “action”: action}:
            print(f’You almost finished the day! Now {action}!’)
        case {“time”: time, “action”: action}:
            print(f’Good {time}! It is time to {action}!’)
        case _:
            print('The time is invalid.')
```

dict

```
class MyClass:
        __match_args__ = ('time', 'action')
        def __init__(self, time, action):
        self.time = time
        self.action = action
match item:
        case MyClass(time = 'evening’, action = ‘relax’):
            print(f'You almost finished the day!’)
        case MyClass(time, action):
            print(f'Good {time}! It is time to {action}!’)
        case _:
            print('The time is invalid.')
```

obj

softserve

match..case

And one of another feature is that we can put conditions to it.

If we are not interested in the parameter, we can again use our “_”

```
match item:
        case [‘evening’, action] if action not in [‘work’,
‘study’]:
                print(f’You almost finished the day! Now
{action}!’)
        case ['evening’, _]:
                print(‘Come on, you deserve some rest!’)
        case [time, action]:
                print(f’Good {time}! It is time to {action}!’)
```

Additional conditions

- Empty list(), tuple(), dict(), string(), and so on return False
- Constant **None** return False
- We can use keyword **in** to check if a value is in a range of values
- We can use operator **is** and **is not** to test for object identity: x is y is true if and only if x and y are the same object. (So, A is B is the same as id(A) == id(B))

```
a = []
if not a:
  print("List is empty")
```

```
keyword = "lambda"
if keyword in ["and", "del", "from", "lambda"]:
    print("{} is a keyword".format(keyword))
```

```
 x = y
>>> id(x)
4401064560
>>> id(y)
4401064560
>>> x is y
True
```

softserve


TO THE FUTURE
softserve