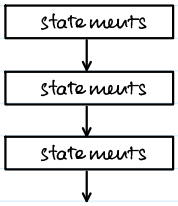
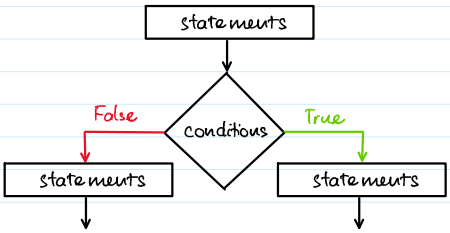
True, False, Truth Value

Tuesday, 25 January 2022 15:09

- Previously, we learned how to make computers follow sequence structure:



Now, we'll try to make computer follow this logic:



- Boolean Type

bool type
- only 2 values True and False
- bool is a sub-type of int, where True == 1, False == 0
  ⇒ They can be added, subtract, ... like int

```
state = False
print(state)
print(type(state))
```

```
False
<class 'bool'>
```

```
print (int(state))
```

```
0
```

NOTE True and False are ||NOT|| strings ⇒ do NOT use 'True' or 'False'

- Truth value

Every Python object has a corresponding truth (Boolean) value

- Use the built-in function `bool()` to determine the ***truth value*** of an object.
- `bool()` will return **`False`** when called on the following values:
  - The number zero (`0`) or (0.0) or (0.00...)
  - Boolean value `False`
  - `None`
  - An empty string, i.e., `''` or `""`
  - An empty list, i.e. `[]`
  - An empty tuple, i.e., `()`
  - An empty dictionary, i.e., `{}`

6

□ Operators □

Tuesday, January 25, 2022 3:19 PM

1. Relational Operators (comparision operators)

Take 2 values (int, float, str) and produce bool value (True or False)

Description	Operator	Example	Result
Less than	<	3<4	True
Greater than	>	3>4	False
Equal to	==	3==4	False
Less than or equal to	<=	3<=4	True
Greater than or equal to	>=	3>=4	False
Not equal to	!=	3!=4	True

Precedence : left to right

NOTE: Python uses "==" for equal because "=" is assignment operation.

Examples :

In [4]:
```
print(1 == 1) #Uses 2 equal signs, i.e. NOT doing variable assignment, but comparing
```
True

In [5]:
```
print(2 > 3)
```
False

In [6]:
```
print(1 != 1)
```
False

In [7]:
```
print(2*3 == 5)
```
False

In [8]:
```
print(2*3 != 5)
```
True

We can, of course, do the same comparisons with variables.

In [10]:
```
x = 5
y = 3
print(x == y)
```
False

In [11]:
```
print(x > y)
```
True

3. All operators precedence

Arithmetic (+, -, /) > Relational (>, ==, !=) > Logical (not, and, or)

```
In [12]: print(not (80 >= 50))
```
False

```
In [13]: print((80 >= 50) and (70 <= 50))
```
False

```
In [14]: print((80 >= 50) or (70 <= 50))
```
True

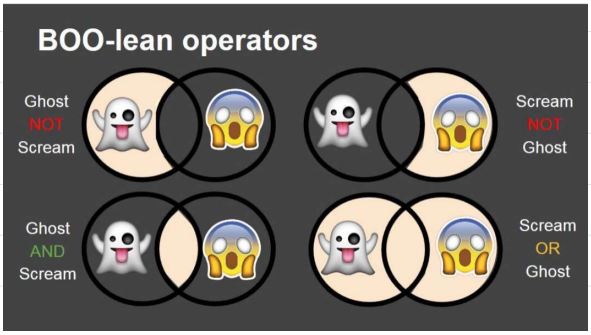
```
In [15]: print(4 > 5 and 2 > 1) #Arithmetic -> Relational -> Logical Operators
```
False

```
In [16]: grade1 = 80
         grade2 = 40
         passed = grade1 >= 50 and grade2 >= 50
         print(passed)
```
False

2. Logical Operators (not, and, or)

Take Boolean operands and evaluate to Boolean values.

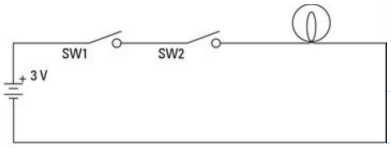
expr1	expr2	expr1 and expr2	expr1 or expr2	not expr1
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True



In specific:

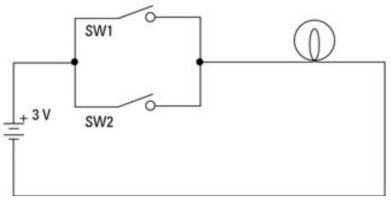
+) The not operator → Results in Boolean value opposite to operand.
→ operands other than bool : return opposite to truth value

+) The and operator → True if both left & right True
False if at least 1 False



→ Operands other than bool: ← False if truth value of 1st operand is False
returns truth value of 2nd operand

+) The or operator → True if at least 1 True
False if both are False



→ Operands other than bool: ← True if truth value of 1st operand is True
returns truth value of 2nd operand

e.g

```
b1 = (" " and "Hello") or 0
b2 = not((" " and 'a') or '') or "from the"
b3 = 'Other' and ('other' or not("OTHER"))
b4 = not(0) and 'side'

print (b1,b2,b3,b4)
Hello from the other side
```

- NOTE Order of Precedence : brackets > not > and > or (should use brackets)

In [22]:
```
print(0 > 5 or "t" > "a" and 'c' != 'd')  #We'll get into why "t" > "a" next class
```
True

In [23]:
```
print(10 < 9 and "t" > "a" and not 'c' == 'd') #not has highest logical operator precedence
```
False

In [24]:
```
print((10 < 9 and "t" > "a") or not "c" == "d")
```
True

NOTE: Lazy Evaluation

Tuesday, January 25, 2022 3:38 PM

Python evaluates 2 operands of logical operations according to the following rule:

① Evaluate left operand to a value and replace operand = value

POSSIBLE SHORT CIRCUIT

② Evaluate right operand to a value and replace operand = value

③ Apply the operator to 2 resultant values

+ Short-circuit evaluation:

+) or: if first operand is True, skip second operand. ⇒ True

+) and: if first operand is False, skip second operand ⇒ False

```
In [25]: def func():
             print("function called")
             return False

In [26]: print(True or func())
         True

In [27]: print(True or print("HELP!"))
         True

In [28]: print(False or func())
         function called
         False

In [29]: print(func() or True)
         function called
         True

In [30]: print(func() or False)
         function called
         False
```

⇒ Python never sees what comes after True or

⊛ We can use this function to create "stoppers" to the statement

```
In [33]: x = 4
         y = 0
         print((x >= 2) and (x/y > 2))
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [33], in <module>
      1 x = 4
      2 y = 0
----> 3 print((x >= 2) and (x/y > 2))

ZeroDivisionError: division by zero
```

We need to check for this case (and do whatever the right thing is). An easy way to do so is to exploit lazy evaluation.

```
In [34]: x = 4
         y = 0
         print((x >=2) and (y != 0) and (x/y > 2))
```

```
False
```

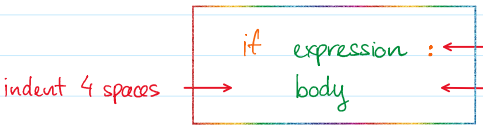
False

Evaluate beforehand whether we can divide by 0

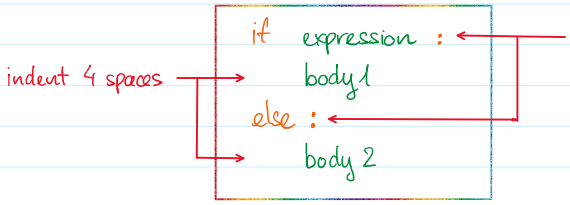
If else Statement

Tuesday, January 25, 2022 3:46 PM

1. The if statement



2. Adding the else statement :



- ONLY one of body 1 or body 2 will execute
  - +) Expression True → execute body 1
  - +) Expression False → execute body 2

⇒ Use if else instead of if if ... to improve efficiency

```
people = 20
cats = 30
dogs = 15

if people < cats:
    print("Too many cats! The world is doomed!")

if people >= cats:
    print("Not many cats! The world is saved!")

if people < dogs:
    print("The world is drooled on!")

if people >= dogs:
    print("The world is dry!")
```

```
Too many cats! The world is doomed!
The world is dry!
```

=

```
people = 20
cats = 30
dogs = 15

if people < cats:
    print("Too many cats! The world is doomed!")
else:
    print("Not many cats! The world is saved!")

if people < dogs:
    print("The world is drooled on!")
else:
    print("The world is dry!")
```

3. Checking if a variable is within Range

In Python, you can check whether a variable is in range in expression

In [37]:
```
x = 3

if 0 < x < 10:
    print(x, "is between 0 and 10")

if 10 > x > 1:
    print(x, "is between 10 and 1")

if 0 <= x <= 3:
    print(x, "is between 0 and 3 inclusive")

if -3 <= x < 0:
    print(x, "is in the interval [-3, 0) inclusive")
else:
    print("or not")
```

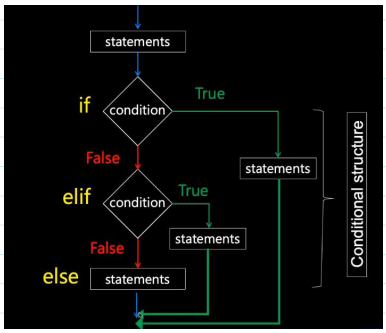
```
3 is between 0 and 10
3 is between 10 and 1
3 is between 0 and 3 inclusive
or not
```

elif (else if) statement

Thursday, January 27, 2022 3:37 PM

```
if condition 1 :
    body 1
elif condition 2 :
    body 2
elif condition N :
    body N
else :
    other_body
```

- NOTE colons and indents!
- Only 1 body is executed.
  - if statement is True, execute body1, exits if structure
  - if statement is False, continue to elif statement
  - elif statement is True, execute elif body, exits if structure
  - elif statement is False, continue to next elif statement
  - All if's and elif's are False, execute else statement



• Multiple if vs elif

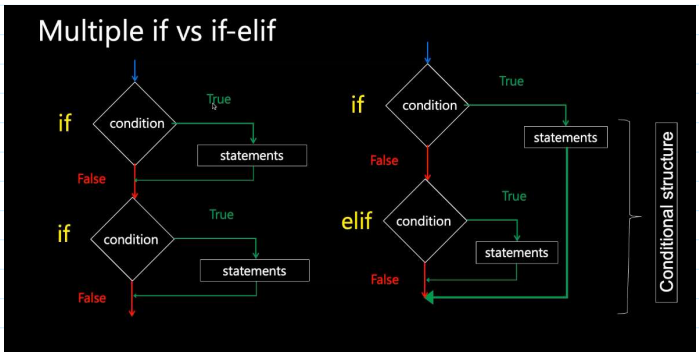
+) Multiple if : if the first if is met, second if still evaluated
+) elif : if the first if is met, elif is NOT evaluated

In [30]:
```
final_grade = 95

if final_grade > 90:
    print('A+')
if final_grade > 80:
    print('A')
if final_grade > 70:
    print('B')
if final_grade > 60:
    print('C')
if final_grade >= 50:
    print('D')
if final_grade < 50:
    print('You failed!')
```

```
A+
A
B
C
D
```

In [2]:
```
final_grade = 95

if final_grade > 90:
    print('A+')
elif final_grade > 80:
    print('A')
elif final_grade > 70:
    print('B')
elif final_grade > 60:
    print('C')
elif final_grade >= 50:
    print('D')
else:
    print('You failed!')
```

```
A+
```



• To prevent program crash by inputting a string, we can add "a layer of protection"

In [50]:
```
final_grade = 'NOMNOMNOM' #a string will crash our program

if 100 >= final_grade >= 90:
    print('A+')
elif 90 > final_grade >= 80:
    print('A')
elif 80 > final_grade >= 70:
    print('B')
elif 70 > final_grade >= 60:
    print('C')
elif 60 > final_grade >= 50:
    print('D')
elif 0 <= final_grade < 50:
    print('You failed!')
else:
    print('Incorrect input')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/mn/8vlvqnvd1gl1d04y6pbyzbhh0000gn/T/ipykernel_56291/274092471.py in <module>
      1 final_grade = 'NOMNOMNOM' #a string will crash our program
      2
----> 3 if 100 >= final_grade >= 90:
      4     print('A+')
      5 elif 90 > final_grade >= 80:
```

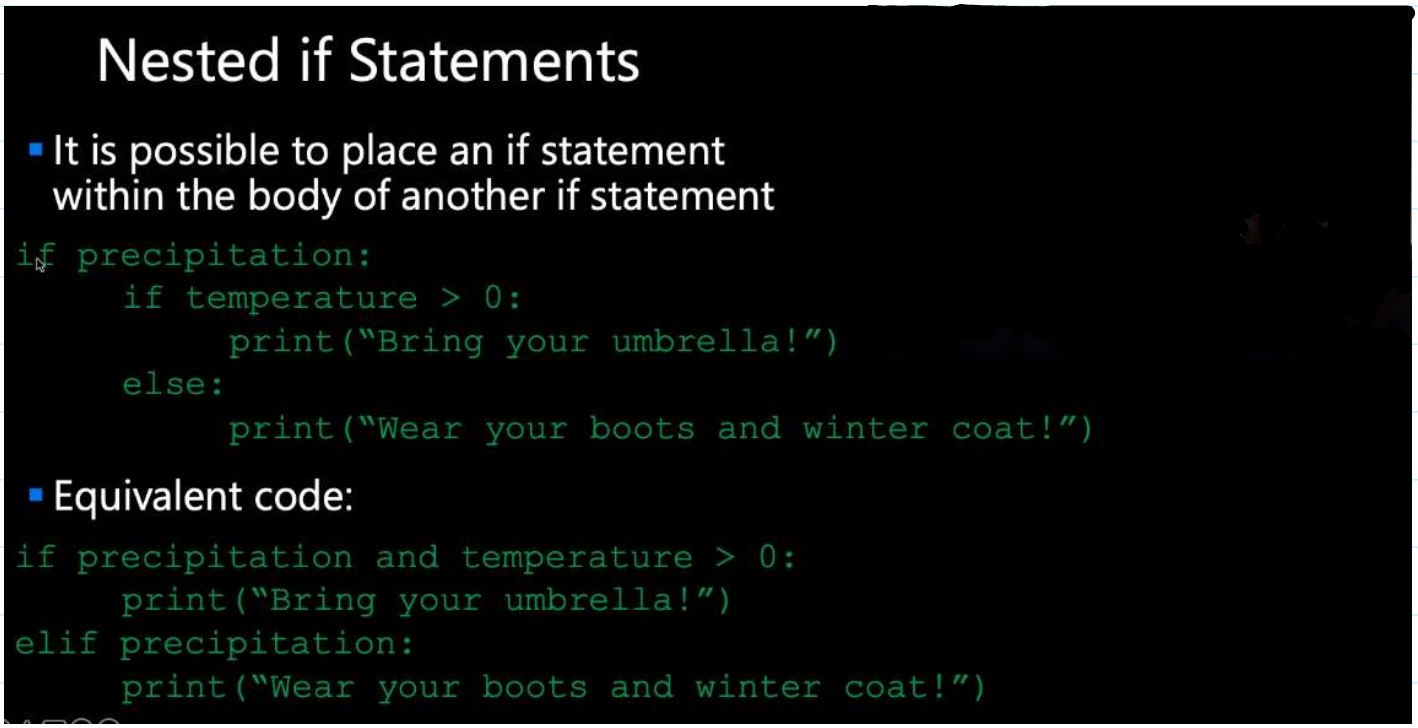
Trying that again... With a new nested if statement

In [49]:
```
final_grade = 'NOMNOMNOM'

if type(final_grade)==int or type(final_grade)==float: #see everything starting to come together?
    if 100 >= final_grade >= 90:
        print('A+')
    elif 90 > final_grade >= 80:
        print('A')
    elif 80 > final_grade >= 70:
        print('B')
    elif 70 > final_grade >= 60:
        print('C')
    elif 60 > final_grade >= 50:
        print('D')
    elif 0 <= final_grade < 50:
        print('You failed!')
    else:
        print('Typo')
else: #else statement isn't necessary, but in this case we can provide a helpful message back to the user
    print('Input not a an integer or float, please try again')
```

```
Input not a an integer or float, please try again
```

Nested if Statements

Thursday, January 27, 2022 3:48 PM



String Comparisons

Thursday, January 27, 2022 3:12 PM

1. String Comparisons

Boolean comparisons can also applied to strings, whether single characters or sets of characters

Description	Operator	Example	Result of example
equality	==	'cat' == 'cat'	True
inequality	!=	'cat' != 'Cat'	True
less than	<	'A' < 'a'	True
greater than	>	'a' > 'A'	True
less than or equal	<=	'a' <= 'a'	True
greater than or equal	>=	'a' >= 'A'	True

When evaluating 2 strings, we are evaluating their numerical values.

RULE:

1. Numbers < CAPITALIZED < lowercase

NOTE:

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
32	[space]	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	[backspace]

2. Evaluate from first character. If '==', move to next. If '!=', stop
If first characters are equal, longer is greater

```
In [7]: print('abracadabra' < 'ace')
True

In [8]: print('abracadabra' > 'ace')
False
```

d

3. CANNOT compare string and int / float

```
In [15]: print('s' <= 3)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/mn/8vlvqnvd1gl1d04y6pbyzbhh0000gn/T/ipykernel_56291/2074115480.py in <module>
----> 1 print('s' <= 3)

TypeError: '<=' not supported between instances of 'str' and 'int'
```

2. ASCII Encoding

ord() → obtain ASCII (integer) representation
chr() → convert integer back to string

3. Test for Substrings

The in operator provides a way to check whether a string appears inside another string
→ Result in Boolean

```
In [23]: print('c' in 'aeiou')
False

In [25]: print('cad' in 'abracadabra')
False

In [26]: print('zoo' in 'ooze')
False
```
← Have to be in order

not in operator returns True if 1st operand is not contained in 2nd operand

is operator returns True if both operands are the same object

is not operator returns True if both operands are not the same object