#4 Truthiness

12/05/2025

- True or false expression is critical in programming to help build conditional logic. and understand the state of an object or expression.
- Boolean data types sole purpose is to capture whether a value is true or false: ~~sole purpose~~
- Boolean values are True and False.
    - → You can print them
    - → Assign them to variables
    - → Pass them around
    - → Test them.

```
print(True)   # True
print(False)  # False
```

```
def make_longer(string, longer):
    if longer:
        return string + string
    else:
        return string

print(make_longer("abc", True))  # 'abcabc'
print(make_longer("xyz", False))  # 'xyz'
```

being passed to parameter longer

```
def is_digit(char):
    if '0' <= char <= '9':
        return True
    else:
        return False

print(is_digit("5"))  # True
print(is_digit("a"))  # False
```

12/05/2025

```
value = True   <- You usually would not do this in
                  real code

if value is True:
    print("It's True")
elif value is False:
    print("It's False")
else:
    print("It's not true or false!")
```

Expressions and Conditions

- Rather than implementing value = True, you would evaluate an expression that evaluates to True or False

```
num = 5

if num < 10:
    print('small number')
else:
    print('large number')

# small number as 5 <10 is True
```

- Functions usually don't return True or False.

```
def is_small(number):
    return number < 10!
num = 15

if is_small(num):
    print("small number")
else:
    print("large number")
```

Prints large number as 15 <10 evaluates to False.

12/05/25

Logical Operators

- Evaluate expression that involve two subexpressions, then return a value that evaluates to True or False

The and operator:

- Evaluates as True when the sub-expressions evaluate as True:

```
print (True and True)    # True
print (True and False)   # False
print (False and True)   # False
print ( False and False) # False
```

```
                       True
num = 5
print((num < 10) and (num > 3))  # True
```

→ parentheses are not essential
→ good for readibility.

- Can chain as many sub-expression as you like with and.

```
          True
num = 5              False                  False
print((num<10) and (num>20) and (num!=5)  # False
```

Evaluation of the expression ended once num > 20 evaluated to false

The or operator:

- evaluates as True when either of the two sub-expressions evaluate as True. False otherwise.

12/05/2025

```
print(True or True) # True
print(False or True) # True
print (True or False) # True
print( False or False) # False
```

The not operator:

- Inverts the truth value of the condition it's applied to. i.e, a True condition will be False and vice versa.

```
print(not True) # False
print (not False) # True

value =3
is_even = (value % 2 ==0)

print(is_even) # False
print (not is_even) # True
```

- not is useful when you want to check the opposite of some condition.

Short-circuit operators:

- when python stops evaluating when it realises the entire expression cannot be true.

For an and operation, ~~print(False~~

- For an or operation, python stops evaluating once it realises an expression cannot be false. i.e, ~~one~~ at least one sub-expression is True.

- Short-circuiting can be dangerous but can also be handy.

12/05/25

```
if name != None and name.isupper():
    print(f"Hi, {name}.")
else:
    print("Hello, whoever you are.")
```

This type of conditional is common.

Truthiness

- Falsy values are:
  - None
  - False
  - Zero numbers: 0, 0.0, 0j
  - Empty strings: ""
  - Empty collections: [], (), {}, set(), frozenset(), range(0)

2.

```
num = 5
if num:
    print("valid number")
else:
    print("error!")
```

5 is truthy → num is not True

```
print(num == True) #False
```