Group 3

IT SPECIALIST: PYTHON

Part 2

HELLO, ALL!
THIS IS GOUP 3

- Agung Prayogi
- Yesaya Arya Danar Kristuadji
- Dian Maulida
- Dwi Fitriawati Fajrin
- Axel Eldrian Hadiwibowo

TABLE OF CONTENTS

LIST, SET, AND STRING OPERATIONS

Python

LEN()

Definition

The len() method will return the sum of the total items in an object. on a string, this function will return the number of characters

Example

The len() method return number 10 because total items in colour variable is 10

```
In [1]: a = [1,3,4,2,7,10]
        len(a)

Out[1]: 6
```

MIN() & MAX()

Definition

The min() and max() functions are to find out what the minimum and maximum values are.

Example

The min() and max() function return number 3 as minimum value of the list and 7 as maximum value of the list.

```
In [3]: a = [3,5,5,7,7,7]
        print(min(a))
        print(max(a))

        3
        7
```

COUNT()

Definition

The count() function is to find out how many times an object appears in the list.

Example

The count() function returns the value 2 because the string "Data Science" contains 2 characters "a".

```
In [6]: b = 'Data Science'
        b.count('a')

Out[6]: 2
```

SORT()

Definition

The sort() method sorts the list ascending by default. This function can also be used by specifying sorting criteria, such as key and reverse.

Example

The sort() function returns data values from lowest (10) to highest (50).

```
In [8]: nilai = [40,20,30,10,50]
        nilai.sort()
        print(nilai)

        [10, 20, 30, 40, 50]
```

SORT PARAMETERS

REVERSE

- Optional.
- reverse=True will sort the list descending. Default is reverse=False

```
In [9]: huruf = ['d','i','a']
        huruf.sort(reverse=True)
        print(huruf)
```

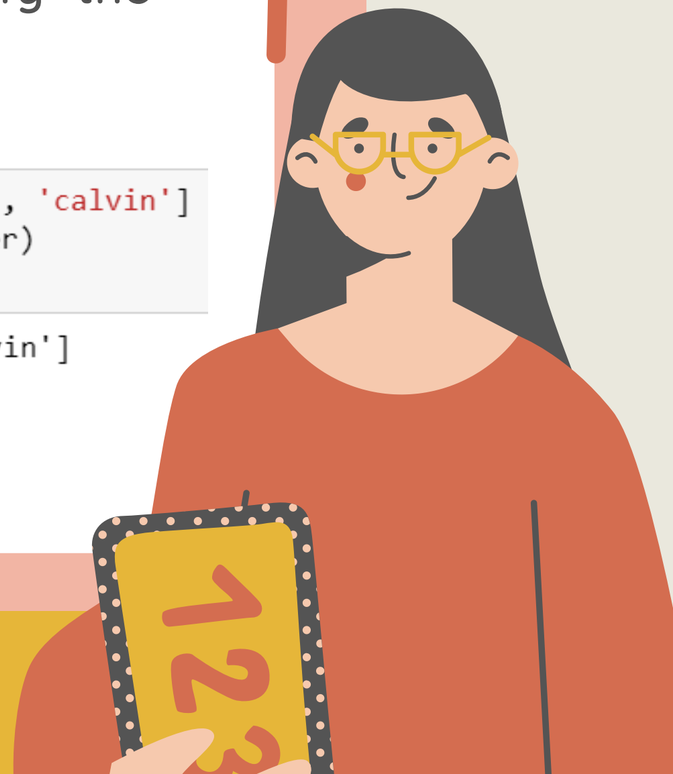
```
['i', 'd', 'a']
```

KEY

- Optional.
- A function to specify the sorting criteria(s)

```
In [10]: nama = ['Budi', 'andi', 'calvin']
         nama.sort(key=str.lower)
         print(nama)
```

```
['andi', 'Budi', 'calvin']
```

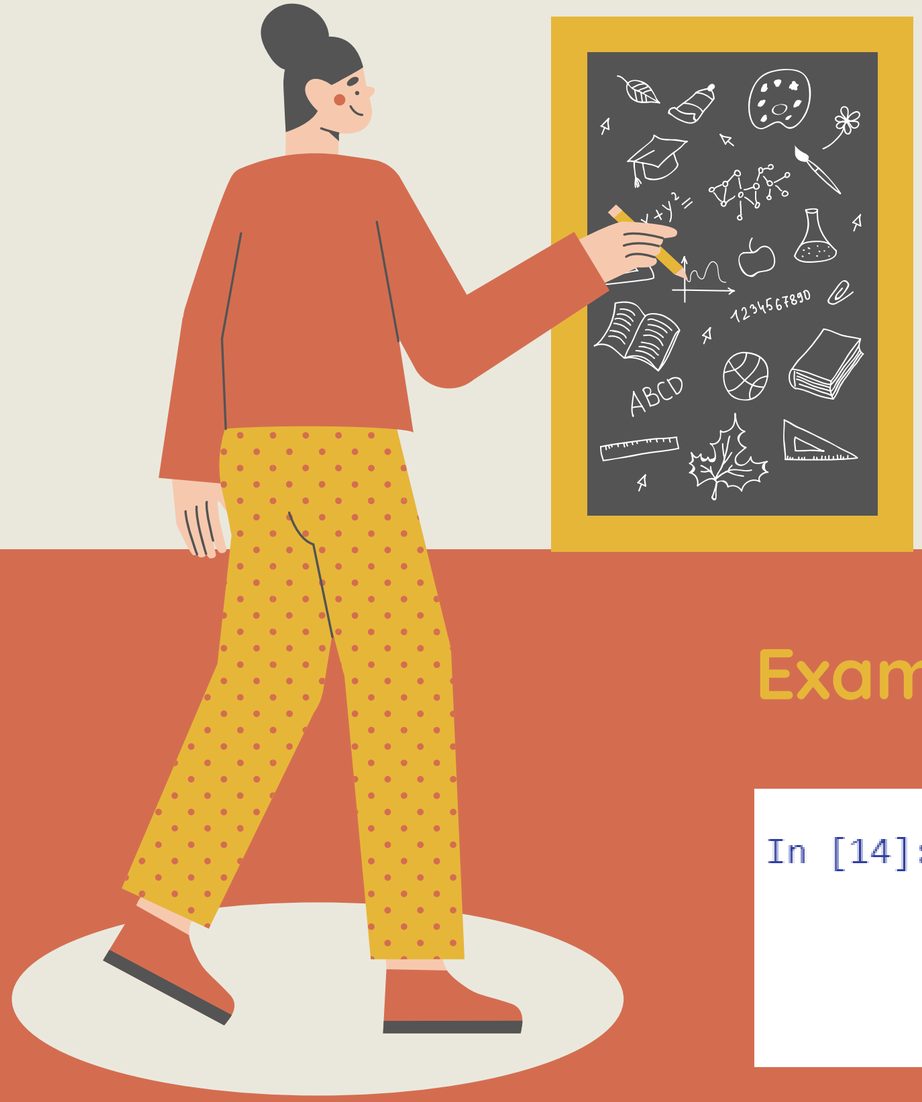


CONCATENATION

Concatenation means joining strings together end-to-end to create a new string use the + operator.

Example:

```
In [11]: a = [3,5,5,7,7,7]
         d = ['d', 'a', 't', 'a']
         a + d

Out[11]: [3, 5, 5, 7, 7, 7, 'd', 'a', 't', 'a']
```

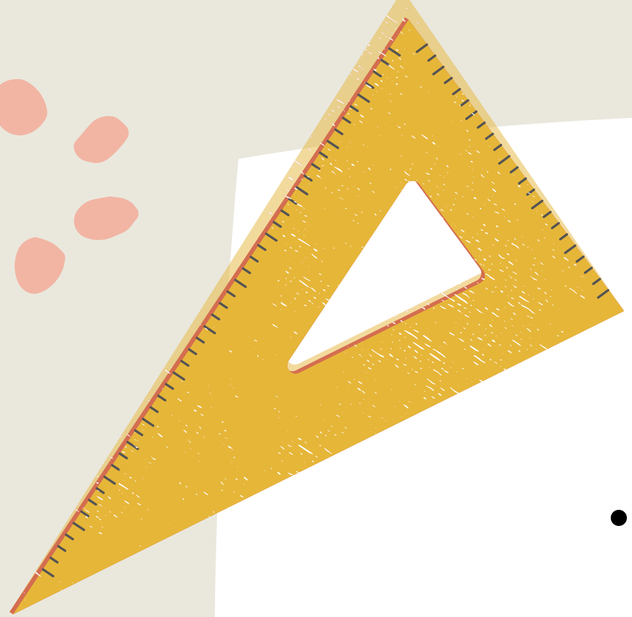


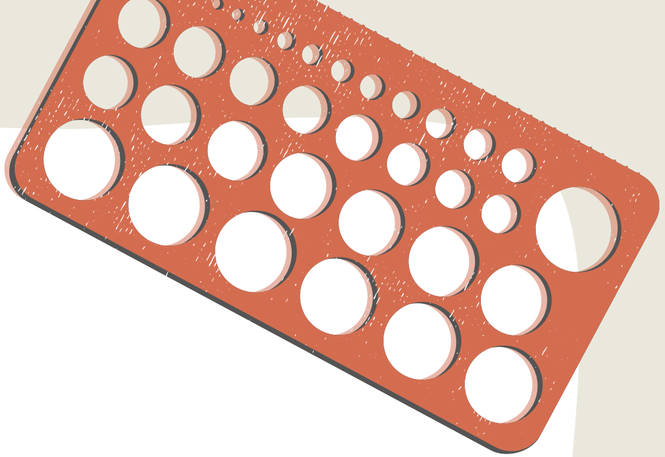
REPLICATION

Replication is multiplication operator (*). When used with one string and one integer, * is the string will repeating a single string however many times the given integer.

Example:

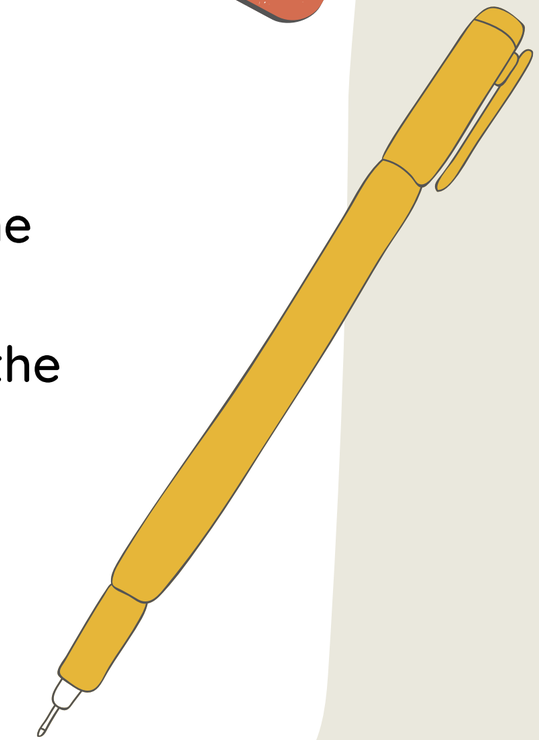
```
In [14]: hewan = ['kucing', 'anjing']
         hewan2 = hewan*3
         print(hewan2)

         ['kucing', 'anjing', 'kucing', 'anjing', 'kucing', 'anjing']
```





RANGE

- The range() function returns a series of numbers with a certain pattern
- To loop (e.g. for) accessing list elements, you can use the range() function in Python
- More details about loop operations will be discussed in the Loop and Loop Control module
- The range function can have 1-3 parameters



RANGE (1 PARAMETER)

Range with 1 parameter n: create a series of numbers starting from 0, as many as n numbers

Example:

```
In [67]: for i in range(3):
             print(i)
```

```
0
1
2
```

RANGE (2 PARAMETERS)

Range with 2 parameters (n,p): create a series of numbers starting from n, until before p (number p is not included)
Often referred to as inclusive n (series start with n) and exclusive p (series do not include p)

Example:

```
In [68]: for i in range (0,3):
             print(i)
```

```
0
1
2
```

RANGE (3 PARAMETERS)

Range with 3 parameters(n,p,q): create a series of numbers starting from n, until before p (the number p is not included), with each element having a difference of q.

Example:

```
In [23]: for i in range (0,6,2):
             print(i)
```

```
0
2
4
```

IN & NOT IN

Operators in and not in: To find out a value or object is in a list. This function will return a boolean value True or False.

Example:

```
In [27]: hari_ini = 'kuliah, studi independen, terus lanjut mengerjakan tugas'

In [28]: 'kuliah' in hari_ini
Out[28]: True

In [29]: 'tugas' not in hari_ini
Out[29]: False
```

ASSIGN VALUES TO MULTIPLE VARIABLES

In assigning values to multiple variables, conventionally, we can mark the variables with the values you want, one at a time.

But, Python has a more practical way, which is that we can assign values to multiple variables at once from a list or tuple element without needing to mark them one by one.

EXAMPLE

```
In [21]: hewan2 = ('sapi', 'jantan', 1)
         print(hewan2)

         ('sapi', 'jantan', 1)

In [22]: nama, jk, usia = hewan2
         print(nama, jk, usia)

         sapi jantan 1
```

In an hewan2 list, there are 3 values in it. We want to define some different variable for each value in the list, so we can do this simultaneously by using a comma (,)

OPERATOR, OPERANDS, AND EXPRESSIONS

Python

ARITHMETIC OPERATORS

+	-	*	/
Addition	Subtraction	Multiplication	Divide
%	**	//	
Modulus	Exponentiation	Floor Division	

Addition (+)

math operation in which we add the numbers together to get their sum

Subtraction (-)

taking something away from a number

Multiplication (*)

to process of calculating the result when a number is taken times

Division (/)

to process of splitting a specific amount into equal parts

```
print("10 + 2 =",10+2)
print("10 - 2 =",10-2)
print("10 * 2 =",10*2)
print("10 / 2 =",10/2)
```

Code

Output

```
10 + 2 = 12
10 - 2 = 8
10 * 2 = 20
10 / 2 = 5.0
```

```
print("10 % 2 =",10%2)
print("10 ** 2 =",10**2)
print("10 // 2 =",10//2)
```

Code

Output

```
10 % 2 = 0
10 ** 2 = 100
10 // 2 = 5
```

Modulus (%)

is the remainder after dividing one number by another

Exponentiation (**)

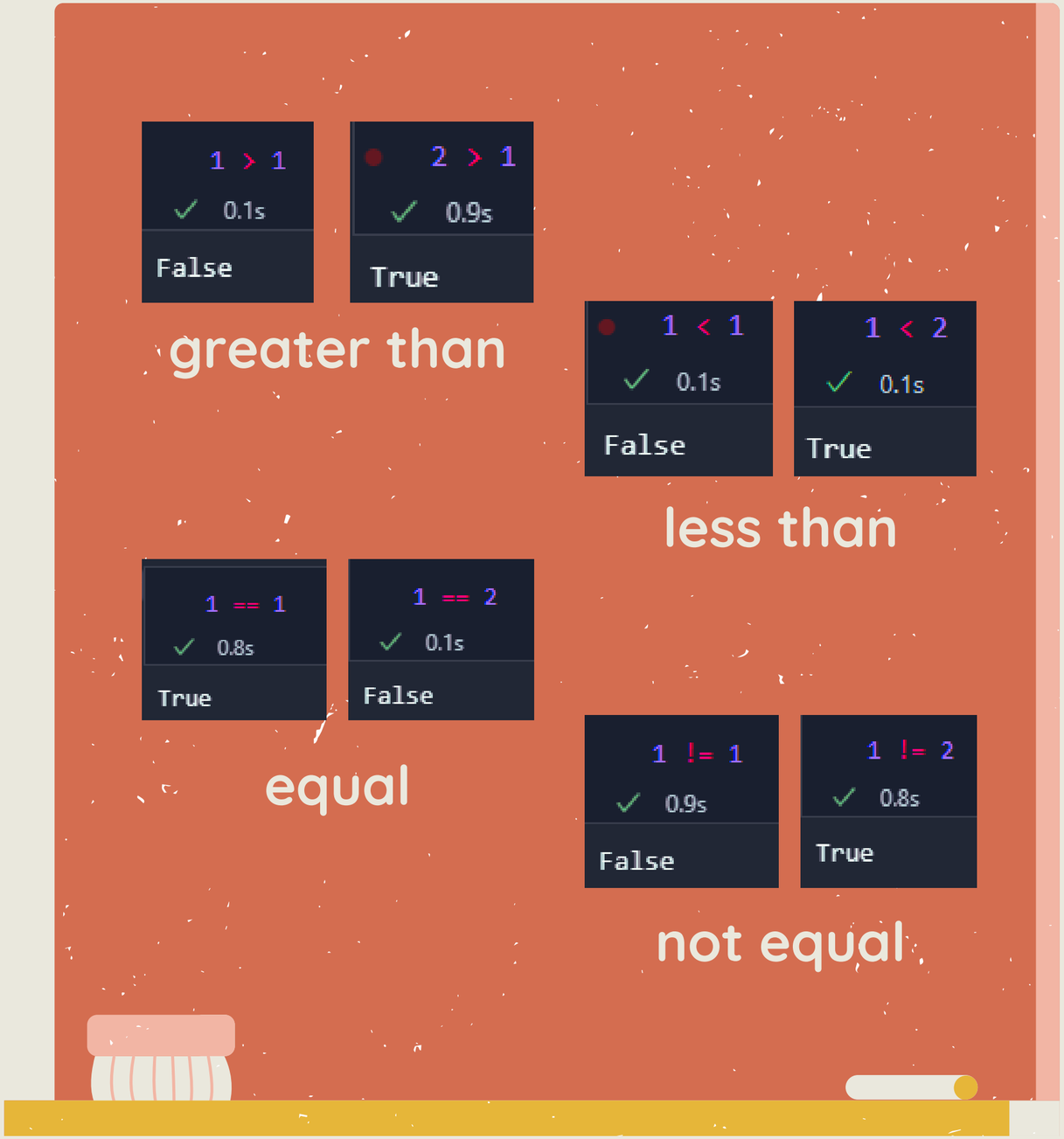
in math is defined as the operation used to represent repeated multiplication

Floor Division (//)

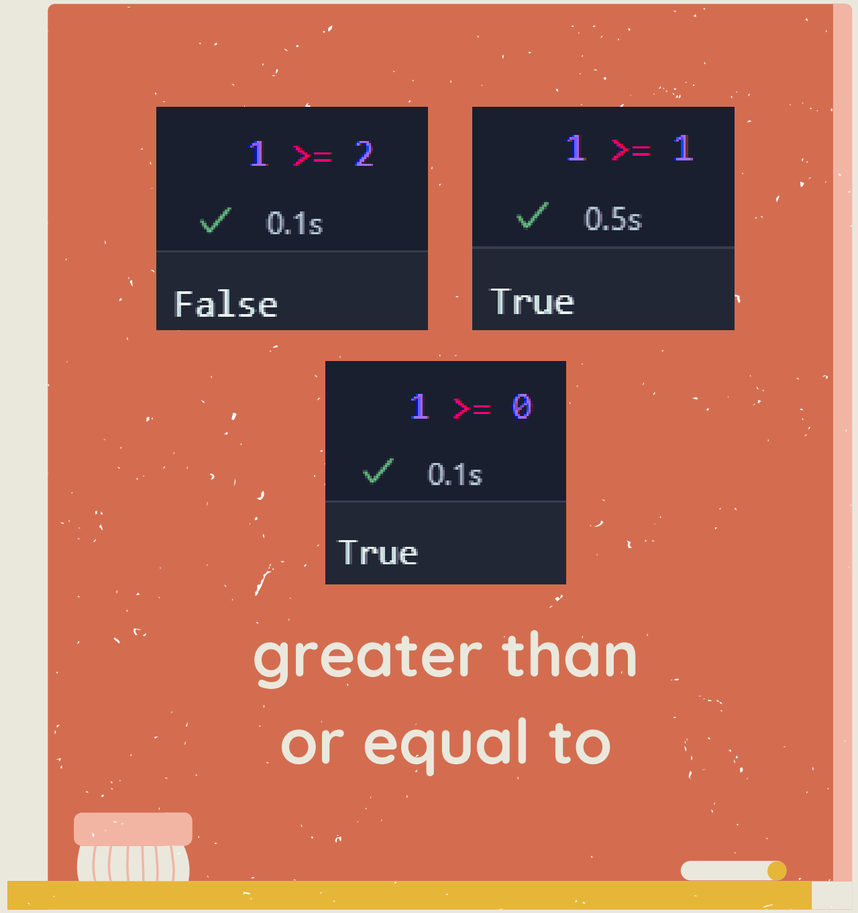
a normal division operation except that it returns the largest possible integer

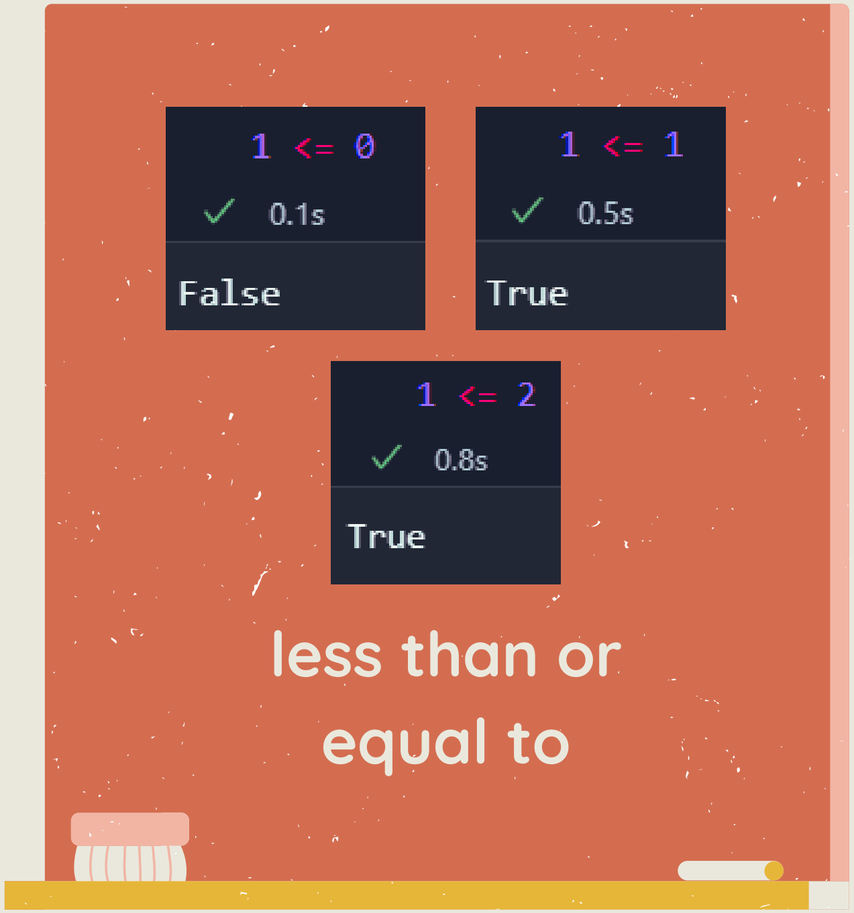
LOGICAL OPERATIONS

>	<	==
>=	<=	!=



LOGICAL OPERATIONS





BOOLEAN

AND

- The 'and' keyword is a logical operator and is used to combine conditional statements
- Both conditions must be fullfilled

OR

- The 'OR' keyword is used in a boolean expression to check that there is at least one true
- If just one side is true the entire expression is true
- Just one condition must be fullfilled

EXAMPLE OF BOOLEAN AND

```
#Medical Test

rontgen = True
blood_test = True
no_rontgen = False
no_blood_test = False
```

```
rontgen and blood_test
```

```
True
```

```
rontgen and no_blood_test
```

```
False
```

```
blood_test and no_rontgen
```

```
False
```

```
no_rontgen and no_blood_test
```

```
False
```

both condition needs to be true so the entire expression can be true

EXAMPLE OF BOOLEAN OR

```
#Medical Test

free_influenza = True
free_covid = True
no_free_influenza = False
no_free_covid = False
```

```
free_influenza or free_covid
```

```
True
```

```
free_influenza or no_free_covid
```

```
True
```

```
free_covid or no_free_influenza
```

```
True
```

```
no_free_influenza or no_free_covid
```

```
False
```

as you can see in this condition, it just needs one statement to be true so the entire expression is true

PYTHON ASSIGNMENT OPERATOR

Assignment Operators are used to assigning values to variables

(=) equal

(+=) Add and Assign

(-=) Subtract AND

(/=) Divide AND

(*=) Multiply AND

(%=) Modulus AND

(**=) Exponent AND

(//=) Floor Division

EXAMPLE OF ASSIGNMENT OPERATOR

```
x = 10
x += 2
print(x)
```

```
12
```

Add and assign

```
x = 10
x -= 2
print(x)
```

```
8
```

Subtract And

```
x = 10
x /= 2
print(x)
```

```
5.0
```

Divide And

```
x = 10
x *= 2
print(x)
```

```
20
```

Multiply And

```
x = 10
x **= 3
print(x)
```

```
1000
```

Exponent And

```
x = 10
x %= 3
print(x)
```

```
1
```

Modulus And

```
x = 10
x //= 3
print(x)
```

```
3
```

Floor Division

PRECEDENCE OPERATOR

Python will always evaluate the arithmetic operators first
(** is highest, then multiplication/ division, then addition/ subtraction)

```
10+5/2
```

```
12.5
```

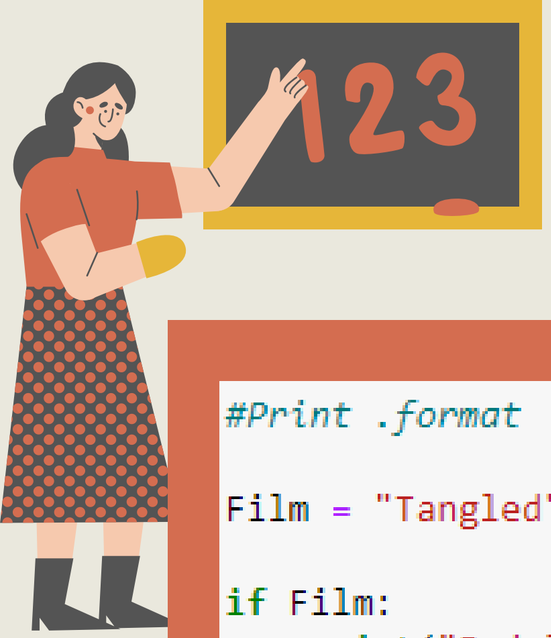
```
#it has the same output if you use ()
#or put the multiply operation first

a = 10 + (5/2)
b = 5/2 + 10
print(a)
print(b)
```

```
12.5
12.5
```

CONDITIONAL EXPRESSION

if

- Only a statement is True will be executed, if it is False it will not
- In Python, the body of the if the indentation indicates statement, the body starts with an indentation and the first unindented line marks the end
- The output is non-zero values as True, none and 0 are interpreted as False



EXAMPLE OF CONDITIONAL EXPRESSION IF

```
#Print .format

Film = "Tangled"

if Film:
    print("Judul film adalah {}". format(Film))
```

```
Judul film adalah Tangled
```

```
#Print comma

Film = "Tangled"

if Film:
    print("Judul film adalah", Film)
```

```
Judul film adalah Tangled
```

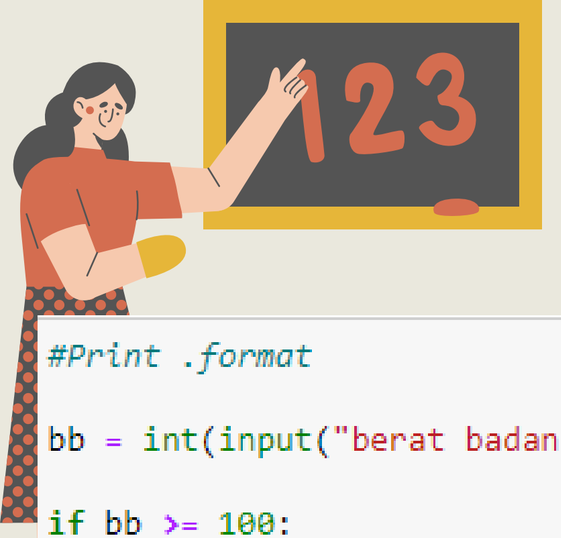
```
#Print specifier

Film = "Tangled"

if Film:
    print("Judul film adalah %s" % (Film))
```

```
Judul film adalah Tangled
```

CONDITIONAL EXPRESSION

else

- The if..else statement evaluates test expression and will execute the body of if only when the test condition is True, if it is False it will not
- 'else' is optional



EXAMPLE OF CONDITIONAL EXPRESSION ELSE

```
#Print .format

bb = int(input("berat badan:"))

if bb >= 100:
    print("berat badanku {} maka aku obesitas". format(bb))
else:
    print("berat badanku {} maka aku tidak obesitas". format(bb))
```

```
berat badan: 123
```

```
#Print .format

bb = int(input("berat badan:"))

if bb >= 100:
    print("berat badanku {} maka aku obesitas". format(bb))
else:
    print("berat badanku {} maka aku tidak obesitas". format(bb))
```

```
berat badan:123
berat badanku 123 maka aku obesitas
```

```
#Print .format

bb = int(input("berat badan:"))

if bb >= 100:
    print("berat badanku {} maka aku obesitas". format(bb))
else:
    print("berat badanku {} maka aku tidak obesitas". format(bb))
```

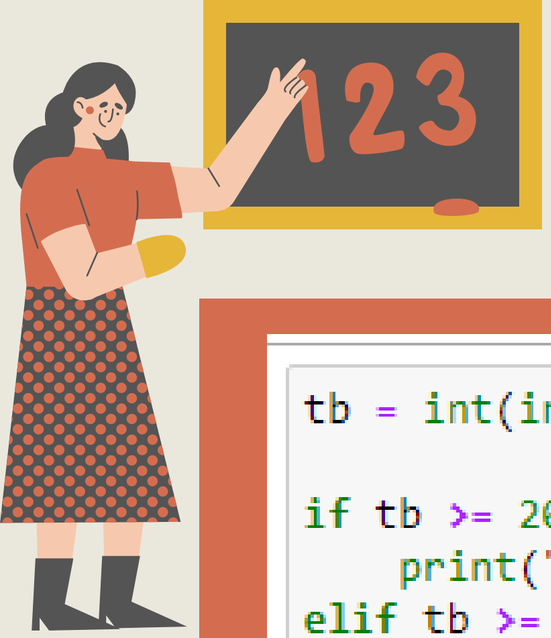
```
berat badan:77
berat badanku 77 maka aku tidak obesitas
```

In this case, if you input 100 or more than 100, you'll get the statement "maka aku obesitas", but if you input number less than 100, you'll get the output "maka aku tidak obesitas"

CONDITIONAL EXPRESSION

elif (else if)

- If the condition for if is False, it checks the condition of the next elif block and so on, if it is False the body of else is executed
- Only one block among the several if...elif...else blocks is executed according to the condition
- In other words, we can say that elif is pythons way of saying "if the previous conditions were not true, then try this condition"



EXAMPLE OF CONDITIONAL EXPRESSION ELIF

```
tb = int(input("tinggi badan:"))

if tb >= 200:
    print("tinggi badan {} masuk tim". format(tb))
elif tb >= 180:
    print("tinggi badan" ,tb, "cadangan 1")
elif tb >= 170:
    print("tinggi badan %d cadangan 2" % (tb))
elif tb >= 160:
    print("tinggi badan {} cadangan 3". format(tb))
else:
    print("tinggi badan {} tidak lolos". format(tb))
```

```
tinggi badan:170
tinggi badan 170 cadangan 2
```

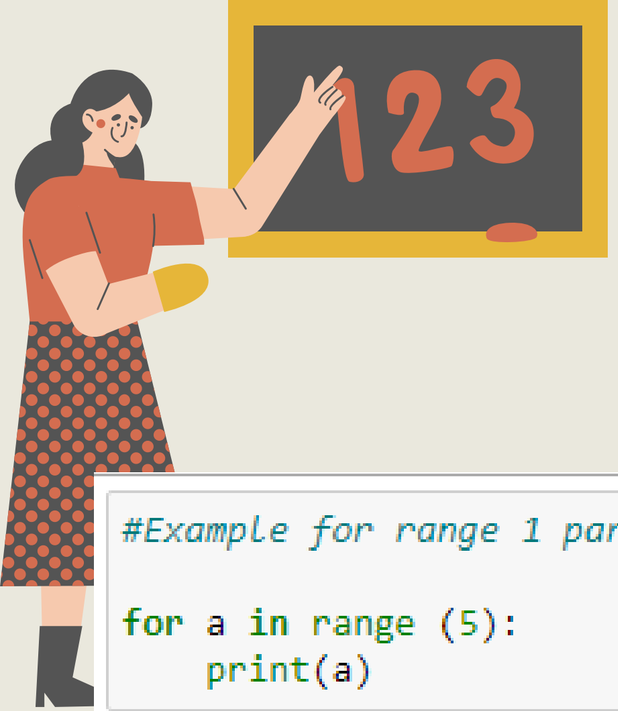
In this case, there are 5 conditions,, if you input greater than or equal to 200 you'll get output "masuk time",
if you input greater than or equal to 180 you'll get the output "cadangan 1"
if you input greater than or equal to 170 you'll get the output "cadangan 2"

etc

FOR LOOPS

- For loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string)

- This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages



EXAMPLE OF FOR LOOPS

```
#Example for range 1 parameter

for a in range (5):
    print(a)
```

```
0
1
2
3
4
```

Loops in range 1 parameter

```
#Example for range 2 parameter

for i in range (8,11):
    print(i)
```

```
8
9
10
```

```
#Example for range 3 parameter

for b in range (20, 28, 3):
    print(b)
```

```
20
23
26
```

Loops in range 2 and 3 parameter

```
#Example for string

for i in 'kampus':
    print("huruf {}". format(i))
```

```
huruf k
huruf a
huruf m
huruf p
huruf u
huruf s
```

Loops in string

```
#Example for List

buah = ['apel', 'mangga', 'melon']

for i in buah:
        print("buah " ,i)
```

```
buah  apel
buah  mangga
buah  melon
```

Loops in List

NESTED LOOPS

- Nested loop is a loop inside a loop
- The "inner loop" will be executed one time for each iteration of the "outer loop"

EXAMPLE OF NESTED LOOP

```
for row in range(4):
    for col in range(4):
        if row == 0 and col <=3:
            print("*", end = " ")
        elif row == 1 and col <= 2:
            print("*", end = " ")
        elif row == 2 and col <= 1:
            print("*", end = " ")
        elif row == 3 and col == 0:
            print("*", end = " ")
    print()
```

```
* * * *
* * *
* *
*
```

In this case, there is a loop inside loop using elif condition

```
color = ['red', 'blue', 'green']
book = ['novel', 'comic', 'dictionary']

for x in color:
  for y in book:
    print(x, y)
```

```
red novel
red comic
red dictionary
blue novel
blue comic
blue dictionary
green novel
green comic
green dictionary
```

WHILE LOOP CONCEPT

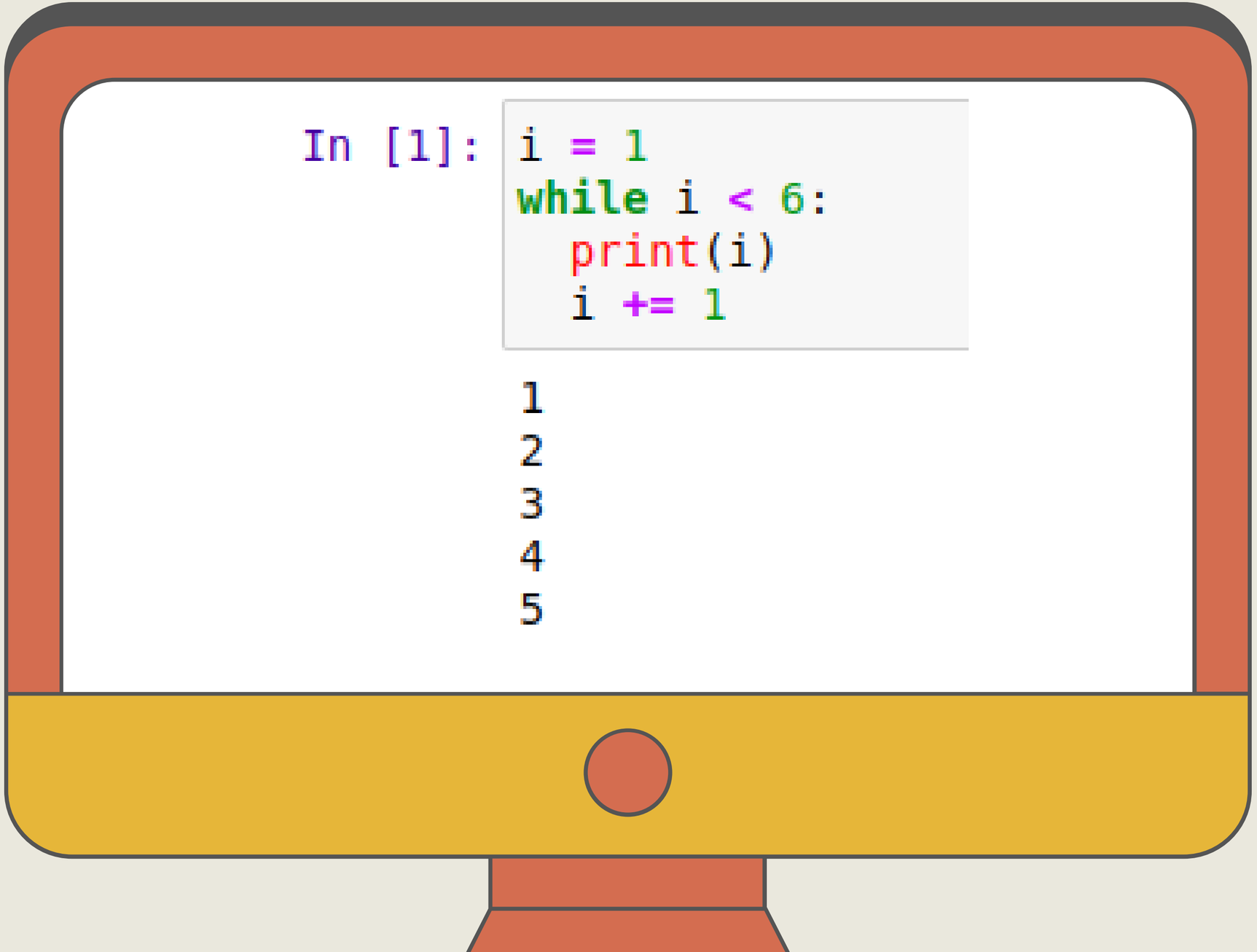
The **while statement** creates a loop that executes a specified statement as long as the test condition evaluates to true. The condition is evaluated before executing the statement

Syntax :

while *condition* :

    statement

EXAMINE THESE EXAMPLES!



LOOP CONTROL STATEMENT

(BREAK, CONTINUE, & PASS)

The **Break** Statement is a loop control statement that is used to terminate the loop

The **continue** statement is used when we want to skip a particular condition and continue the rest execution

The **pass** statement is used as a placeholder for future code. When it's executed, nothing happen, but you avoid getting an error

WHILE USING BREAK STATEMENT

In [1]:
```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

```
1
2
3
```

WHILE USING CONTINUE STATEMENT

In [1]:
```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

```
1
2
4
5
6
```

WHILE USING PASS STATEMENT

In [2]:
```
i = 0
while i < 6:
  i += 1
  if i % 5 == 0:
    pass
  print(i)
```

```
1
2
3
4
5
6
```

LIST COMPREHENSION

Create a list with inline loops and if statement

Python

LIST COMPREHENSION IS A WAY TO GENERATE NEW LISTS BASED ON PRE-EXISTING LISTS OR ITERABLES

EXAMPLE LIST COMPREHENSION

In [1]:
```
number = [2,3,4]
```

In [2]:
```
kubik = []
```

In [3]:
```
for i in number:
    kubik.append(i**3)
print(kubik)
```

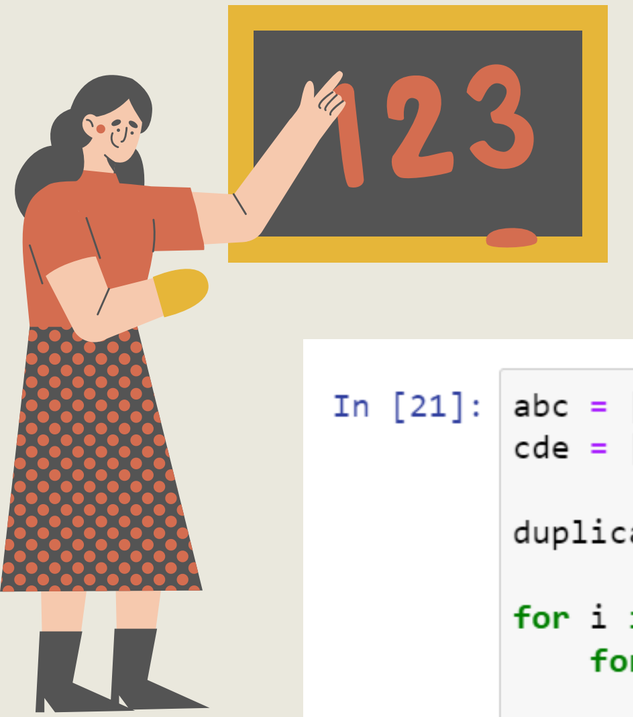
```
[8, 27, 64]
```

OTHER WAY TO CREATE LIST COMPREHENSION

Syntax

```
new_list = [expression for_loop_one_or_more_conditions)
```

```
In [4]: angka = [2,3,4]

In [6]: kubik = [i**3 for i in angka]

In [14]: print(kubik)
         [8, 27, 64]
```



EXAMPLE OF LIST COMPREHENSION

```
In [21]: abc = ['a','b','c']
         cde = ['c','d','e']

         duplicate=[]

         for i in abc:
             for j in cde:
                 if i == j:
                     duplicate.append(i)
         print(duplicate)

         ['c']
```

Example 01

```
In [11]: number = [5,6,7,9,11]

In [12]: modulo = [i%2 for i in number]

In [19]: print(modulo)

         [1, 0, 1, 1, 1]
```

Example 02

```
In [8]:  number = [5,6,7,9,11]

In [9]:  floor_division = []

In [10]: for i in number:
             floor_division.append(i//2)
         print(floor_division)

         [2, 3, 3, 4, 5]
```

Example 03

NOTE !

- Underscore (underscore) including valid variable naming.
- In general "_" can be used as a throwaway variable (variable is not important).

```
In [7]: python = ['PYTHON','PROGRAMMING']

        lower = [_.lower() for _ in python]

        print(lower)

        ['python', 'programming']
```

TYPES OF ERROR IN PYTHON PROGRAMMING LANGUAGE

Python

TYPES OF ERROR BASED ON OCCURRENCE

Syntax Error

Syntax errors occur when Python can't understand what we mean.

Exceptions

Errors that occur while the process is in progress are called exceptions and can be fatal if not handled

SYNTAX ERROR

```
In [2]: for i in 'data':
        print(i)

          Input In [2]
            print(i)
            ^
        IndentationError: expected an indented block
```

The placement of indentance (space at the beginning) is inappropriate.

SYNTAX ERROR

```
In [3]: for i in 'data'
            print(i)

  Input In [3]
    for i in 'data'
                    ^
SyntaxError: invalid syntax
```

After the condition of the "for" command required a colon.

EXCEPTION

```
In [4]: food = ('Nasi Goreng','Soto Ayam Kampung','Pecel Ponorogo','Bakso Ngalam')

        food.append('Bakmi Goreng')

        print(food)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [4], in <cell line: 3>()
      1 food = ('Nasi Goreng','Soto Ayam Kampung','Pecel Ponorogo','Bakso Ngalam')
----> 3 food.append('Bakmi Goreng')
      5 print(food)

AttributeError: 'tuple' object has no attribute 'append'
```

Tuple doesn't have attribute 'append'.

https://github.com/eldrians/python-guide-2

THANK YOU