Keywords

- **Python keywords** are unique words reserved with defined meanings and functions that we can only apply for those functions. You'll never need to import any keyword into your program because they're permanently present.
- Python's built-in methods and classes are not the same as the keywords. Built-in methods and classes are constantly present; however, they are not as limited in their application as keywords.
- Assigning a particular meaning to Python keywords means you can't use them for other purposes in our code. You'll get a message of SyntaxError if you attempt to do the same. If you attempt to assign anything to a built-in method or type, you will not receive a SyntaxError message; however, it is still not a smart idea.

❖ List of Python keywords

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

❖ True, False, None Keyword

- **True:** This keyword is used to represent a boolean true. If a statement is true, “True” is printed.
- **False:** This keyword is used to represent a boolean false. If a statement is false, “False” is printed.
- **None:** This is a special constant used to denote a null value or a void. It’s important to remember, 0, any empty container(e.g. empty list) does not compute to None.

```
print( 4 == 4 )
print( 6 > 9 )
print( True or False )
```

Output:

```
True
False
True
```

❖ Operator Keywords: and, or, not, in, is

- Several Python keywords are employed as operators to perform mathematical operations.
- In many other computer languages, these operators are represented by characters such as &, |, and!. All of these are keyword operations in Python:

Mathematical Operations	Operations in Other Languages	Python Keyword
AND, ∧	&&	and
OR, ∨	||	or
NOT, ¬	!	not
CONTAINS, ∈		in
IDENTITY	===	is

Example:

```
x = 10
y = 3

addition = x + y
comparison = x > y
logical_and = x > 5 and y < 5
assignment = x += y

print("Addition:", addition)
print("Comparison:", comparison)
print("Logical AND:", logical_and)
print("Assignment:", assignment)
```

Output:

```
Addition: 13
Comparison: True
Logical AND: False
SyntaxError: cannot use assignment expressions with subscript
```

❖ Iteration Keywords – for, while, break, continue

- **for:** This keyword is used to control flow and for looping.
- **while:** Has a similar working like “for”, used to control flow and for looping.
- **break:** “break” is used to control the flow of the loop. The statement is used to break out of the loop and passes the control to the statement following immediately after loop.
- **continue:** “continue” is also used to control the flow of code. The keyword skips the current iteration of the loop but does not end the loop.

Examples:

```
# Using the "for" loop to iterate through a list
numbers = [1, 2, 3]
for num in numbers:
    print(num)
# Using the "while" loop for conditional iteration
count = 0
while count < 3:
    print("Count:", count)
    count += 1
```

Output:

```
1
2
3


0
1
2
```

❖ Conditional keywords – if, else, elif

- **if :** It is a control statement for decision making. **Truth expression forces control to go in "if" statement block.**
- **else** : It is a control statement for decision making. **False expression forces control to go in "else" statement block.**
- **elif** : It is a control statement for decision making. It is short for "**else if**"

Examples:

```
# Checking if a number is positive
number = 10


if number > 0:
    print("The number is positive.")
```

Output:

```
The number is positive.
```

❖ Return Keywords – Return, Yield

- **return :** This keyword is used to return from the function.
- **yield :** This keyword is used like return statement but is used to return a generator.

❖ Exception Handling Keywords - try, except, raise, finally, and assert

- **try:** This keyword is designed to handle exceptions and is used in conjunction with the keyword except to handle problems in the program. When there is some kind of error, the program inside the "try" block is verified, but the code in that block is not executed.
- **except:** As previously stated, this operates in conjunction with "try" to handle exceptions.
- **finally:** Whatever the outcome of the "try" section, the "finally" box is implemented every time.
- **raise:** The raise keyword could be used to specifically raise an exception.
- **assert:** This method is used to help in troubleshooting. Often used to ensure that code is correct. Nothing occurs if an expression is interpreted as true; however, if it is false, "AssertionError" is raised. An output with the error, followed by a comma, can also be printed.