Keywords in python

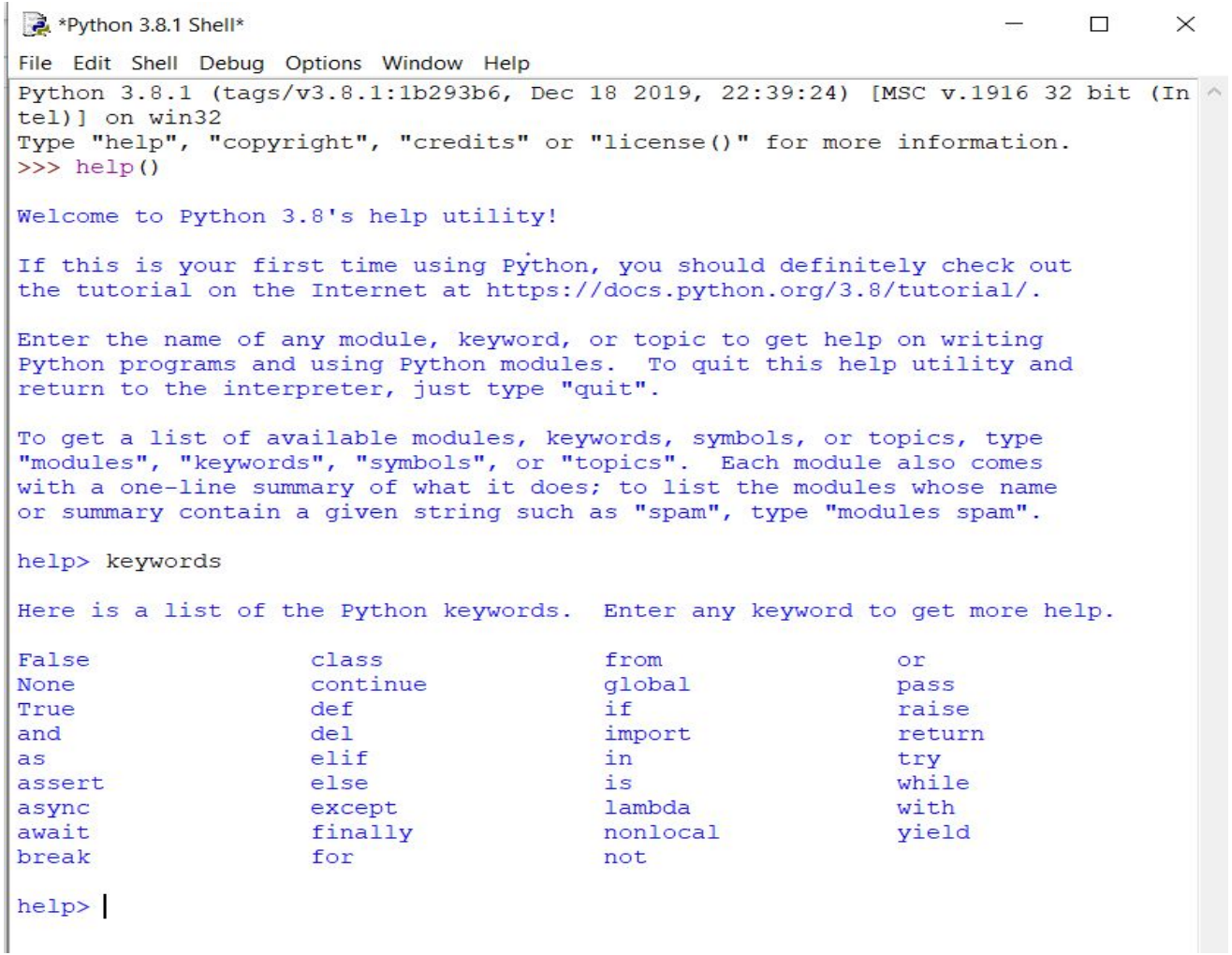
By Bhadra Reddy

Python keywords/reserved keywords

- Python keywords are the reserved words
- They are used by python interpreter to understand the program
- Keywords define the structure of programs
- We can’t use keywords to name program entities such as variables, class, and functions.

How many keywords in python?

- Python has a lot of keywords. The number keeps on growing with the new features coming in python.
- We can get the complete list of keywords using python interpreter help utility.


Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> help()
Welcome to Python 3.8's help utility!
If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.8/tutorial/.
Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".
To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".
help> keywords
Here is a list of the Python keywords. Enter any keyword to get more help.
False class from or
None continue global pass
True def if raise
and del import return
as elif in try
assert else is while
async except lambda with
await finally nonlocal yield
break for not
help>

Conclusion

- Python keywords have specific functions.
- They are used by the python interpreter to understand the code and execute them.
- There are 35 keywords in Python. The number will keep on growing with new features.

Brief Introduction of keywords

Serial No	Keyword	Description	Example
1	False	instance of class bool.	x = False
2	class	keyword to define a class.	class Foo: pass
3	from	clause to import class from module	from collections import OrderedDict
4	or	Boolean operator	x = True or False
5	None	instance of NoneType object	x = None

6	continue	continue statement, used in the nested for and while loop. It continues with the next cycle of the nearest enclosing loop.	numbers = range(1,11) for number in numbers: if number == 7: continue
7	global	global statement allows us to modify the variables outside the current scope.	x = 0 def add(): global x x = x + 10 add() print(x) # 10
8	pass	Python pass statement is used to do nothing. It is useful when we require some statement but we don't want to execute any code.	def foo(): pass
9	True	instance of bool class.	x = True
10	def	keyword used to define a function.	def bar(): print("Hello")

11	if	if statement is used to write conditional code block.	x = 10 if x%2 == 0: print("x is even") # prints "x is even"
12	raise	The raise statement is used to throw exceptions in the program.	def square(x): if type(x) is not int: raise TypeError("Require int argument") print(x * x)
13	and	Boolean operator for and operation.	x = True y = Falseprint(x and y) # False
14	del	The del keyword is used to delete objects such as variables, list, objects, etc.	s1 = "Hello" print(s1) # Hello del s1 print(s1) # NameError: name 's1' is not defined
15	import	The import statement is used to import modules and classes into our program.	# importing class from a module from collections import OrderedDict# import module import math

16	return	The return statement is used in the function to return a value.	def add(x,y): return x+y
17	as	Python as keyword is used to provide name for import, except, and with statement.	from collections import OrderedDict as od import math as mwith open('data.csv') as file: pass # do some processing on filetry: pass except TypeError as e: pass
18	elif	The elif statement is always used with if statement for "else if" operation.	x = 10if x > 10: print('x is greater than 10') elif x > 100: print('x is greater than 100') elif x == 10: print('x is equal to 10') else: print('x is less than 10')
19	in	Python in keyword is used to test membership.	l1 = [1, 2, 3, 4, 5]if 2 in l1: print('list contains 2')s = 'abcd'if 'a' in s: print('string contains a')
20	try	Python try statement is used to write exception handling code.	x = '' try: i = int(x) except ValueError as ae: print(ae)# invalid literal for int() with base 10: ''

21	assert	The assert statement allows us to insert debugging assertions in the program. If the assertion is True, the program continues to run. Otherwise AssertionError is thrown.	def divide(a, b): assert b != 0 return a / b
22	else	The else statement is used with if-elif conditions. It is used to execute statements when none of the earlier conditions are True.	if False: pass else: print('this will always print')
23	is	Python is keyword is used to test if two variables refer to the same object. This is same as using == operator.	fruits = ['apple'] fruits1 = ['apple'] f = fruits print(f is fruits) # True print(fruits1 is fruits) # False
24	while	The while statement is used to run a block of statements till the expression is True.	i = 0 while i < 3: print(i) i+=1# Output # 0 # 1 # 2

25	async	New keyword introduced in Python 3.5. This keyword is always used in couroutine function body. It's used with asyncio module and await keywords.	import asyncio import timeasync def ping(url): print(f'Ping Started for {url}') await asyncio.sleep(1) print(f'Ping Finished for {url}')async def main(): await asyncio.gather(ping('askpython.com'), ping('python.org'),)if __name__ == '__main__': then = time.time() loop = asyncio.get_event_loop() loop.run_until_complete(main()) now = time.time() print(f'Execution Time = {now – then}')# Output Ping Started for askpython.com Ping Started for python.org Ping Finished for askpython.com Ping Finished for python.org Execution Time = 1.004091739654541
26	await	New keyword in Python 3.5 for asynchronous processing.	Above example demonstrates the use of async and await keywords.
27	lambda	The lambda keyword is used to create lambda expressions.	multiply = lambda a, b: a * b print(multiply(8, 6)) # 48

28	with	Python with statement is used to wrap the execution of a block with methods defined by a context manager. The object must implement __enter__() and __exit__() functions.	with open('data.csv') as file: file.read()
29	except	Python except keyword is used to catch the exceptions thrown in try block and process it.	Please check the try keyword example.
30	finally	The finally statement is used with try-except statements. The code in finally block is always executed. It's mainly used to close resources.	def division(x, y): try: return x / y except ZeroDivisionError as e: print(e) return -1 finally: print('this will always execute')print(division(10, 2)) print(division(10, 0))# Output this will always execute 5.0 division by zero this will always execute -1

31	nonlocal	The nonlocal keyword is used to access the variables defined outside the scope of the block. This is always used in the nested functions to access variables defined outside.	def outer(): v = ‘outer’def inner(): nonlocal v v = ‘inner’inner() print(v)outer()
32	yield	Python yield keyword is a replacement of return keyword. This is used to return values one by one from the function.	def multiplyByTen(*kwargs): for i in kwargs: yield i * 10a = multiplyByTen(4, 5,) # a is generator object, an iterator# showing the values for i in a: print(i)# Output 40 50

33	break	The break statement is used with nested “for” and “while” loops. It stops the current loop execution and passes the control to the start of the loop.	number = 1 while True: print(number) number += 2 if number > 5: break print(number) # never executed# Output 1 3 5
34	for	Python for keyword is used to iterate over the elements of a sequence or iterable object.	s1 = ‘Hello’ for c in s1: print(c)# Output H e l l o
35	not	The not keyword is used for boolean not operation.	x = 20 if x is not 10: print(‘x is not equal to 10’)x = True print(not x) # False