

Python Keywords - An Introduction

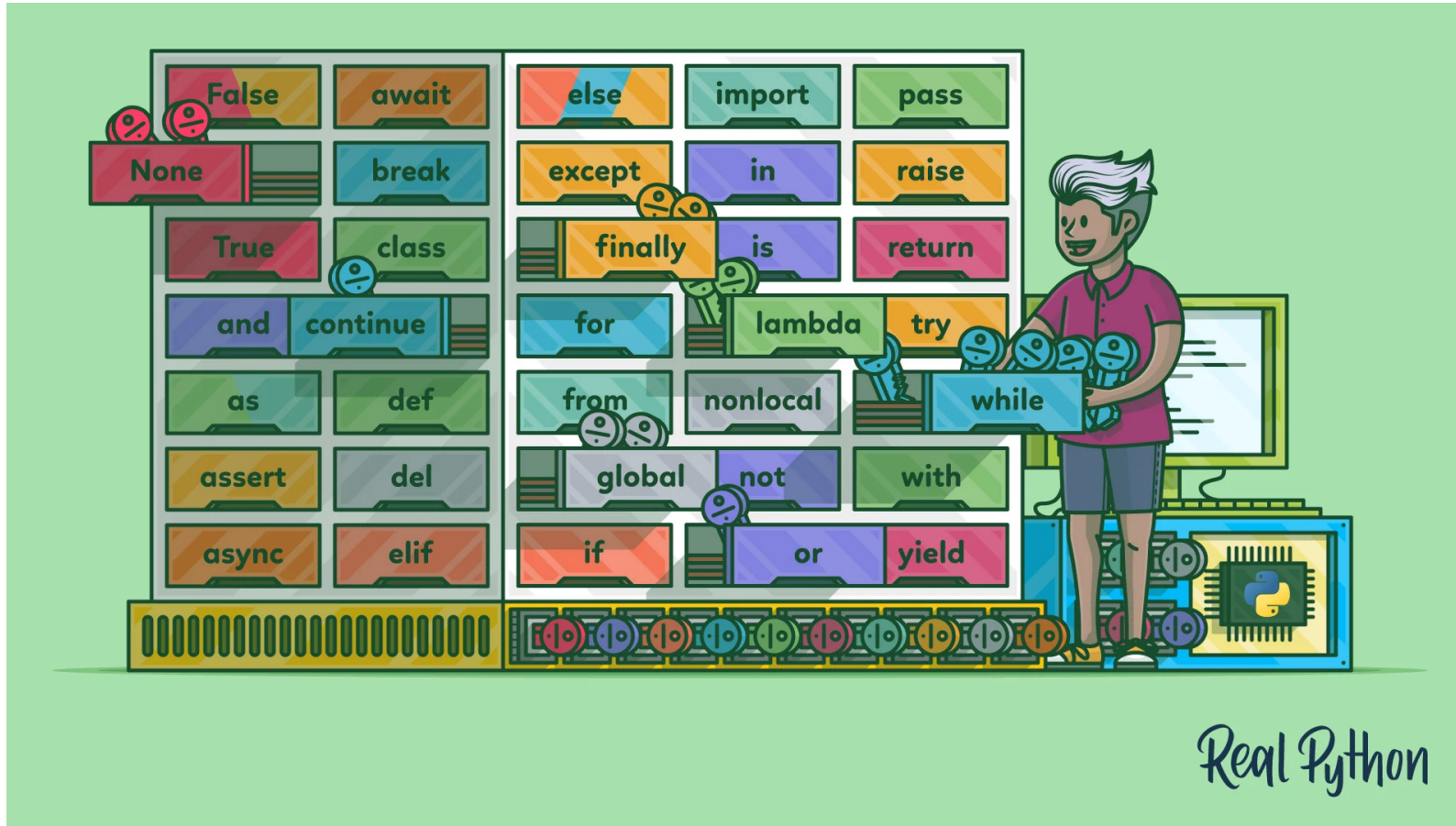




Table of Contents

1. Listing All Python Keywords
2. Understanding Keywords
3. Categorizing Keywords
4. Identifying Keywords
5. Deprecated Keywords

Table of Contents

- ▶ 1. Listing All Python Keywords
- 2. Understanding Keywords
- 3. Categorizing Keywords
- 4. Identifying Keywords
- 5. Deprecated Keywords

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

Soft Keywords

match	case	—
-------	------	---

Next Up

1. Listing All Python Keywords
- ▶ 2. **Understanding Keywords**
3. Categorizing Keywords
4. Identifying Keywords
5. Deprecated Keywords

Table of Contents

1. Listing All Python Keywords
- ▶ 2. **Understanding Keywords**
3. Categorizing Keywords
4. Identifying Keywords
5. Deprecated Keywords

Keywords:

- ✓ always available
- ✓ fundamental building blocks
- ! special reserved words
- ! restrictive in their usage

Keywords:

- ✓ always available
- ✓ fundamental building blocks
- ! special reserved words
- ! restrictive in their usage

Soft Keywords:

- ✓ always available
- ✓ fundamental building blocks

Keywords can change over time

Before Python 3: `and`, `del`, `for`, `is`, `raise`, `assert`, `elif`, `from`, `lambda`, `return`,
`break`, `else`, `global`, `not`, `try`, `class`, `except`, `if`, `or`, `while`, `continue`, `exec`,
`import`, `pass`, `def`, `finally`, `in`, `print`

Python 3.0: `True`, `False`, `None`, `as`, `with`, ~~`print`~~, ~~`exec`~~

Python 3.5: `async`, `await`

Python 3.10: `match`, `case`, `_`

Next Up

1. Listing All Python Keywords
2. Understanding Keywords
- ▶ 3. **Categorizing Keywords**
4. Identifying Keywords
5. Deprecated Keywords

Table of Contents

1. Listing All Python Keywords
2. Understanding Keywords
- ▶ 3. **Categorizing Keywords**
4. Identifying Keywords
5. Deprecated Keywords

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

Soft Keywords

match	case	_
-------	------	---

Python Keywords



Category	Keywords
Value	True, False, None
Control Flow	if, elif, else
Operator	and, or, not, in, is
Iteration	for, while, break, continue, else
Structure	def, class, with, as, pass, lambda
Returning	return, yield
Importing	import, from, as
Exception-Handling	try, except, raise, finally, else, assert
Asynchronous	async, await
Scope Handling	del, global, nonlocal
Soft Keywords	match, case, _



Grab Your Cheat Sheet!



Python Keywords



Category	Keywords
Value	True, False, None
Control Flow	if, elif, else
Operator	and, or, not, in, is
Iteration	for, while, break, continue, else
Structure	def, class, with, as, pass, lambda
Returning	return, yield
Importing	import, from, as
Exception-Handling	try, except, raise, finally, else, assert
Asynchronous	async, await
Scope Handling	del, global, nonlocal
Soft Keywords	match, case, _

Check out the *Supporting Materials* below 

Value Keywords

- True
- False
- None

```
work = False  
weekend = True  
to_do = None
```


Control Flow Keywords

- `if`
- `elif`
- `else`

```
if user_input == "YES":  
    print("... proceeding.")  
elif user_input == "NO":  
    print("... aborted.")  
else:  
    print("Options: YES/NO")
```

```
debug = True if js_debug_bool == "true" else False
```

Operator Keywords

- and
- or
- not
- in
- is

```
require_login = user is None or verification_needed(user)
```

```
if message and not silent_mode:  
    print(message)
```

```
if filename in special_files:  
    print(f"... special handling for {filename}")
```

Iteration Keywords

- for
- while
- break
- continue
- else

```
import random

secret_number = random.randint(1, 99)

while guess := input("Guess my random number (1-99)"):
    if not guess.isnumeric():
        continue

    number = int(guess)
    if number == secret_number:
        break
    elif number < secret_number:
        print(f"{number} is too low")
    else:
        print(f"{number} is too high")
else:
    print("You gave up!")

print(f"The number was {secret_number}")
```

Structure Keywords

- def
- class
- with
- as
- pass
- lambda

```
def scream(input_str):  
    print(f"{input_str.upper()}!")
```

```
class CustomError(Exception):  
    pass
```

```
with open('data.txt', 'r') as f:  
    data = f.read()
```

```
ids = ["1", "3", "10", "2"]  
sorted_ids = sorted(ids, key=lambda id_str: int(id_str))
```

Returning Keywords

- return
- yield

```
def instruction_as_text(steps):  
    return "\n".join(steps)
```

```
def instruction_step(steps):  
    yield from steps
```

Importing Keywords

- import
- from
- as

```
import os
```

```
from datetime import date, time, datetime  
from time import time as t
```

Exception-Handling Keywords

- try
- except
- raise
- finally
- else
- assert

```
def items_per_person(num_items, num_person):  
    assert num_items > 0  
  
    try:  
        items_per_person = num_items / num_person  
    except ZeroDivisionError:  
        print("There are no people :(")  
        raise  
    else:  
        print(f"Each person gets {items_per_person} items.")  
    finally:  
        print(f"Shared {num_items} items with {num_person} person.")  
    return items_per_person
```

Asynchronous Keywords

- `async`
- `await`

```
import asyncio
import socket
from http.client import HTTPConnection

async def check_host_connectivity(url):
    connection = HTTPConnection(url)
    try:
        connection.request("HEAD", "/")
        return f"{url} is online!"
    except socket.error:
        return f"{url} is not online!"
    finally:
        connection.close()

async def asynchronous_check(urls):
    async def _check(url):
        response = await check_host_connectivity(url)
        print(response)

    await asyncio.gather(*(_check(url) for url in urls))

urls = ["www.python.org", "www.real-python.com"]
asyncio.run(asynchronous_check(urls))
```


Scope Handling Keywords

- `del`
- `global`
- `nonlocal`

```
all_numbers = []

def mean():
    total = 0
    length = 0
    def _mean(number):
        global all_numbers
        nonlocal total, length
        total += number
        length += 1
        all_numbers.append(number)
        return total / length
    return _mean
```

```
del response["headers"]
del response["errors"]
```

Soft Keywords

- match
- case
- _

```
def sum_ingredients(ingredient_counts):  
    match ingredient_counts:  
        case []:  
            return 0  
        case [first, *rest]:  
            return first + sum_ingredients(rest)  
        case _:  
            raise ValueError(f"Can only sum lists.")
```

Next Up

1. Listing All Python Keywords
2. Understanding Keywords
3. Categorizing Keywords
- ▶ 4. **Identifying Keywords**
5. Deprecated Keywords

Table of Contents

1. Listing All Python Keywords
2. Understanding Keywords
3. Categorizing Keywords
- ▶ 4. **Identifying Keywords**
5. Deprecated Keywords

Next Up

1. Listing All Python Keywords
2. Understanding Keywords
3. Categorizing Keywords
4. Identifying Keywords
- ▶ 5. **Deprecated Keywords**

Table of Contents

1. Listing All Python Keywords
2. Understanding Keywords
3. Categorizing Keywords
4. Identifying Keywords
- ▶ 5. **Deprecated Keywords**

Deprecated Keywords

- `print`
- `exec`

The `print` statement has been replaced with a `print()` function, with keyword arguments to replace most of the special syntax of the old `print` statement.

Removed keyword: `exec()` is no longer a keyword; it remains as a function.

– <https://docs.python.org/3.0/whatsnew/3.0.html>

Summary

1. Listing All Python Keywords
2. Understanding Keywords
3. Categorizing Keywords
4. Identifying Keywords
5. Deprecated Keywords

Summary

▶ 1. Listing All Python Keywords

2. Understanding Keywords
3. Categorizing Keywords
4. Identifying Keywords
5. Deprecated Keywords

- `keyword.kwlist`
- `keyword.sftkwlist`
- `help("keywords")`

Summary

1. Listing All Python Keywords

▶ 2. **Understanding Keywords**

3. Categorizing Keywords

4. Identifying Keywords


5. Deprecated Keywords

Keywords are:

- special reserved words
- restrictive in their usage
- fundamental building blocks
- always available

Summary

1. Listing All Python Keywords
2. Understanding Keywords
- ▶ 3. **Categorizing Keywords**
4. Identifying Keywords
5. Deprecated Keywords

Python Keywords	
	
Category	Keywords
Value	True, False, None
Control Flow	if, elif, else
Operator	and, or, not, in, is
Iteration	for, while, break, continue, else
Structure	def, class, with, as, pass, lambda
Returning	return, yield
Importing	import, from, as
Exception-Handling	try, except, raise, finally, else, assert
Asynchronous	async, await
Scope Handling	del, global, nonlocal
Soft Keywords	match, case, _



Summary

1. Listing All Python Keywords
2. Understanding Keywords
3. Categorizing Keywords
- ▶ 4. **Identifying Keywords**
5. Deprecated Keywords

- Use:
 - `keyword.iskeyword()`
 - `keyword.issoftkeyword()`
 - `help()`
- Leverage Syntax Highlighting
- Get an `SyntaxError`

Summary

1. Listing All Python Keywords
2. Understanding Keywords
3. Categorizing Keywords
4. Identifying Keywords
- ▶ 5. **Deprecated Keywords**

Since Python 3.0:

- `exec` ☒ `exec()`
- `print` ☒ `print()`

What's Next?

- Have a closer look at Keywords
 - `lambda` : <https://realpython.com/python-lambda/>
 - `yield` : <https://realpython.com/introduction-to-python-generators/>
 - `is` : <https://realpython.com/python-is-identity-vs-equality/>
 - `pass` : <https://realpython.com/python-pass/>
 - `import` : <https://realpython.com/python-import/>

What's Next?

- Refresh your knowledge
 - <https://realpython.com/python-conditional-statements/>
 - <https://realpython.com/python-boolean/>
 - <https://realpython.com/python-for-loop/>
 - <https://realpython.com/python-while-loop/>

What's Next?

- Investigate Soft Keywords
 - `match`, `case`: <https://realpython.com/python310-new-features/>

What's Next?

- Use Keywords in their context
 - <https://realpython.com/defining-your-own-python-function/>
 - <https://realpython.com/working-with-files-in-python/>
 - <https://realpython.com/python3-object-oriented-programming>

