What's New in Python

출시 버전 ***3.8.18***

A. M. Kuchling

2월 08, 2024

Contents

편집자 Raymond Hettinger

이 기사에서는 파이썬 3.7과 비교하여 3.8의 새로운 기능에 관해 설명합니다. 자세한 내용은 changelog를 참조하세요.

Python 3.8 was released on October 14th, 2019.

1 요약 – 배포 주요 사항

2 새로운 기능

2.1 대입 표현식

더 큰 표현식의 일부로 변수에 값을 대입하는 새로운 문법 := 이 있습니다. 바다코끼리의 눈과 엄니를 닮아서 "바다코끼리 연산자(the walrus operator)"라고 친근하게 알려져 있습니다.

이 예에서, 대입 표현식은 `len()`을 두 번 호출하지 않도록 합니다:

```
if (n := len(a)) > 10:
    print(f"List is too long ({n} elements, expected <= 10)")
```

정규식 일치 중에도 비슷한 이점이 있습니다. 일치 객체가 두 번 필요합니다, 일치하는지 검사할 때 한 번, 서브 그룹을 추출할 때 한 번:

```
discount = 0.0
if (mo := re.search(r'(\d+)% discount', advertisement)):
    discount = float(mo.group(1)) / 100.0
```

이 연산자는 루프 종료를 검사하기 위해 값을 계산한 다음 루프의 바디에서 그 값이 다시 필요한 while 루프에도 유용합니다:

```
# Loop over fixed length blocks
while (block := f.read(256)) != '':
    process(block)
```

필터링 조건에서 계산된 값이 표현식 바디에도 필요한 리스트 컴프리헨션에서 또 다른 사용 사례가 생깁니다:

```
[clean_name.title() for name in names
 if (clean_name := normalize('NFC', name)) in allowed_names]
```

복잡성을 줄이고 가독성을 개선하는 명확한 사례로 바다코끼리 연산자 사용을 제한하십시오.

자세한 설명은 **PEP 572**를 참조하십시오.

(Contributed by Emily Morehouse in bpo-35224.)

2.2 위치 전용 매개 변수

일부 함수 매개 변수를 위치적으로 지정해야만 하고 키워드 인자로 사용할 수 없도록 지시하는 새로운 함수 매개 변수 문법 / 이 있습니다. 이것은 Larry Hastings의 Argument Clinic 도구로 어노테이트된 C 함수들에 대해 `help()`가 보여주는 것과 같은 표기법입니다.

다음 예에서, 매개 변수 *a*와 *b*는 위치 전용이며, *c*나 *d*는 위치나 키워드일 수 있으며, *e*나 *f*는 키워드 전용이어야 합니다:

```
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)
```

다음은 유효한 호출입니다:

```
f(10, 20, 30, d=40, e=50, f=60)
```

하지만, 다음은 잘못된 호출입니다:

```
f(10, b=20, c=30, d=40, e=50, f=60)   # b cannot be a keyword argument
f(10, 20, 30, 40, 50, f=60)           # e must be a keyword argument
```

이 표기법의 한 가지 사용 사례는 순수 파이썬 함수가 기존 C 코드 함수의 동작을 완전히 흉내 낼 수 있다는 것입니다. 예를 들어, 내장 `divmod()` 함수는 키워드 인자를 허용하지 않습니다:

```
def divmod(a, b, /):
    "Emulate the built in divmod() function"
    return (a // b, a % b)
```

또 다른 사용 사례는 매개 변수 이름이 도움이 되지 않을 때 키워드 인자를 배제하는 것입니다. 예를 들어, 내장 `len()` 함수의 서명은 `len(obj, /)`입니다. 이것은 다음과 같은 어색한 호출을 금지합니다:

```
len(obj='hello')  # The "obj" keyword argument impairs readability
```

매개 변수를 위치 전용으로 표시하면 클라이언트 코드를 손상할 위험 없이 매개 변수 이름을 나중에 변경할 수 있다는 추가적인 이점이 있습니다. 예를 들어, `statistics` 모듈에서, 매개 변수 이름 *dist*는 나중에 변경될 수 있습니다. 이것은 다음과 같은 함수 명세 때문에 가능해졌습니다:

```
def quantiles(dist, /, *, n=4, method='exclusive')
    ...
```

/의 왼쪽에 있는 매개 변수는 가능한 키워드로 노출되지 않기 때문에, 매개 변수 이름은 `**kwargs`에서 계속 사용할 수 있습니다:

```
>>> def f(a, b, /, **kwargs):
...     print(a, b, kwargs)
...
>>> f(10, 20, a=1, b=2, c=3)         # a and b are used in two ways
10 20 {'a': 1, 'b': 2, 'c': 3}
```

이는 임의의 키워드 인자를 받아들여야 하는 함수와 메서드의 구현을 크게 단순화합니다. 예를 들어, 다음은 `collections` 모듈의 코드에서 뽑아온 것입니다:

```
class Counter(dict):

    def __init__(self, iterable=None, /, **kwds):
        # Note "iterable" is a possible keyword argument
```

자세한 설명은 **PEP 570**을 참조하십시오.

(Contributed by Pablo Galindo in bpo-36540.)

2.3 컴파일된 바이트 코드 파일을 위한 병렬 파일 시스템 캐시

새 PYTHONPYCACHEPREFIX 설정(-X pycache_prefix로도 사용 가능합니다)은 각 소스 디렉터리 내의 기본 __pycache__ 하위 디렉터리 대신 별도의 병렬 파일 시스템 트리를 사용하도록 묵시적 바이트 코드 캐시를 구성합니다.

캐시의 위치는 sys.pycache_prefix로 보고됩니다 (None은 __pycache__ 하위 디렉터리의 기본 위치를 나타냅니다).

(Contributed by Carl Meyer in bpo-33499.)

2.4 디버그 빌드는 릴리스 빌드와 같은 ABI를 사용합니다.

파이썬은 이제 릴리스나 디버그 모드 중 어느 것으로 빌드되더라도 같은 ABI를 사용합니다. 유닉스에서, 파이썬이 디버그 모드로 빌드될 때, 이제 릴리스 모드로 빌드된 C 확장과 안정(stable) ABI를 사용해서 빌드된 C 확장을 로드할 수 있습니다.

릴리스 빌드와 디버그 빌드는 이제 ABI 호환됩니다: Py_DEBUG 매크로를 정의하는 것은 더는 Py_TRACE_REFS 매크로를 암시하지 않습니다, 이것은 ABI 비 호환성만 도입할 뿐입니다. sys.getobjects() 함수와 PYTHONDUMPREFS 환경 변수를 추가하는 Py_TRACE_REFS 매크로는 새로운 ./configure --with-trace-refs 빌드 옵션을 사용하여 설정할 수 있습니다. (Contributed by Victor Stinner in bpo-36465.)

유닉스에서, C 확장은 안드로이드와 Cygwin을 제외하고는 더는 libpython에 링크되지 않습니다. 이제 정적으로 링크된 파이썬이 공유 라이브러리 파이썬을 사용하여 빌드된 C 확장을 로드할 수 있습니다. (Contributed by Victor Stinner in bpo-21536.)

유닉스에서, 파이썬이 디버그 모드로 빌드될 때, 임포트는 이제 릴리스 모드로 컴파일된 C 확장과 안정(stable) ABI로 컴파일된 C 확장도 찾습니다. (Contributed by Victor Stinner in bpo-36722.)

파이썬을 응용 프로그램에 내장하려면, 새로운 --embed 옵션을 python3-config --libs --embed에 전달하여 -lpython3.8을 얻어야 합니다 (응용 프로그램을 libpython에 링크합니다). 3.8 이하를 모두 지원하려면, 먼저 python3-config --libs --embed를 시도하고, 실패하면 python3-config --libs(--embed 없이)로 대체하십시오.

파이썬을 응용 프로그램에 내장하기 위해, pkg-config python-3.8-embed 모듈을 추가했습니다: pkg-config python-3.8-embed --libs 는 -lpython3.8을 포함합니다. 3.8 이하를 모두 지원하려면, 먼저 pkg-config python-X.Y-embed --libs 를 시도하고, 실패하면 pkg-config python-X.Y --libs(--embed 없이)로 대체하십시오 (X.Y를 파이썬 버전으로 교체하십시오).

반면에, pkg-config python3.8 --libs 는 더는 -lpython3.8을 포함하지 않습니다. C 확장은 libpython에 링크되어서는 안 됩니다 (안드로이드와 Cygwin은 예외인데, 이들은 스크립트로 처리됩니다); 이 변경은 의도적으로 이전 버전과 호환되지 않습니다. (Contributed by Victor Stinner in bpo-36721.)

2.5 f-문자열은 스스로 설명하는 표현식과 디버깅을 위해 =를 지원합니다.

f-문자열에 = 지정자를 추가했습니다. f'{expr=}'과 같은 f-문자열은 표현식의 텍스트, 등호, 평가된 표현식의 표현(repr)으로 확장됩니다. 예를 들어:

```
>>> user = 'eric_idle'
>>> member_since = date(1975, 7, 31)
>>> f'{user=} {member_since=}'
"user='eric_idle' member_since=datetime.date(1975, 7, 31)"
```

일반적인 f-문자열 포맷 지정자를 사용하면 표현식의 결과가 표시되는 방식을 더 잘 제어할 수 있습니다:

```
>>> delta = date.today() - member_since
>>> f'{user=!s}  {delta.days=:,d}'
'user=eric_idle  delta.days=16,075'
```

= 지정자는 계산을 표시할 수 있도록 전체 표현식을 표시합니다:

```
>>> print(f'{theta=}  {cos(radians(theta))=:.3f}')
theta=30  cos(radians(theta))=0.866
```

(Contributed by Eric V. Smith and Larry Hastings in bpo-36817.)

2.6 PEP 578: 파이썬 런타임 감사 훅

이 PEP는 감사 훅(Audit Hook)과 확인된 열기 훅(Verified Open Hook)을 추가합니다. 둘 다 파이썬과 네이티브 코드에서 사용 가능해서, 순수 파이썬 코드로 작성된 응용 프로그램과 프레임워크가 추가 알림을 활용할 수 있도록 함과 동시에 임베더(embedder)나 시스템 관리자가 감사가 항상 활성화된 파이썬 빌드를 배치할 수 있도록 합니다.

자세한 설명은 **PEP 578**을 참조하십시오.

2.7 PEP 587: 파이썬 초기화 구성

PEP 587은 파이썬 초기화를 구성하는 새로운 C API를 추가하여 전체 구성에 대한 세밀한 제어와 개선된 에러 보고를 제공합니다.

새로운 구조체:

- `PyConfig`
- `PyPreConfig`
- `PyStatus`
- `PyWideStringList`

새로운 함수:

- `PyConfig_Clear()`
- `PyConfig_InitIsolatedConfig()`
- `PyConfig_InitPythonConfig()`
- `PyConfig_Read()`
- `PyConfig_SetArgv()`
- `PyConfig_SetBytesArgv()`
- `PyConfig_SetBytesString()`
- `PyConfig_SetString()`
- `PyPreConfig_InitIsolatedConfig()`
- `PyPreConfig_InitPythonConfig()`
- `PyStatus_Error()`
- `PyStatus_Exception()`
- `PyStatus_Exit()`
- `PyStatus_IsError()`
- `PyStatus_IsExit()`
- `PyStatus_NoMemory()`
- `PyStatus_Ok()`
- `PyWideStringList_Append()`
- `PyWideStringList_Insert()`

- `Py_BytesMain()`
- `Py_ExitStatusException()`
- `Py_InitializeFromConfig()`
- `Py_PreInitialize()`
- `Py_PreInitializeFromArgs()`
- `Py_PreInitializeFromBytesArgs()`
- `Py_RunMain()`

이 PEP는 이러한 내부 구조체에 `_PyRuntimeState.preconfig`(`PyPreConfig` 형)와 `PyInterpreterState.config` (`PyConfig` 형) 필드를 추가합니다. `PyInterpreterState.config`는 전역 구성 변수와 기타 내부(private) 변수를 대체하는 새로운 참조 구성이 됩니다.

설명서는 파이썬 초기화 구성을 참조하십시오.

자세한 설명은 **PEP 587**을 참조하십시오.

(Contributed by Victor Stinner in bpo-36763.)

2.8 Vectorcall: a fast calling protocol for CPython

The "vectorcall" protocol is added to the Python/C API. It is meant to formalize existing optimizations which were already done for various classes. Any extension type implementing a callable can use this protocol.

이것은 현재 잠정적(provisional)입니다. 목표는 파이썬 3.9에서 완전히 공개하는 것입니다.

자세한 설명은 **PEP 590**을 참조하십시오.

(Contributed by Jeroen Demeyer and Mark Shannon in bpo-36974.)

2.9 아웃 오브 밴드 데이터 버퍼를 사용하는 피클 프로토콜 5

멀티 코어나 멀티 머신 프로세싱을 활용하기 위해 `pickle`을 사용해서 파이썬 프로세스 간에 큰 데이터를 전송할 때, 메모리 복사를 줄이고 데이터 종속적 압축과 같은 사용자 정의 기술을 적용하여 전송을 최적화하는 것이 중요합니다.

`pickle` 프로토콜 5는 통신 계층의 재량에 따라 **PEP 3118** 호환 데이터가 주 피클 스트림과 별도로 전송될 수 있는 아웃 오브 밴드 버퍼를 지원합니다.

자세한 설명은 **PEP 574**를 참조하십시오.

(Contributed by Antoine Pitrou in bpo-36785.)

3 기타 언어 변경

- `continue` 문은 구현 문제로 인해 `finally` 절에서 사용할 수 없었습니다. 파이썬 3.8에서는 이 제한이 제거되었습니다. (Contributed by Serhiy Storchaka in bpo-32489.)
- `bool`, `int` 및 `fractions.Fraction` 형에는 이제 `float`와 `decimal.Decimal`에서 발견되는 것과 유사한 `as_integer_ratio()` 메서드가 있습니다. 이 작은 API 확장을 통해 `numerator, denominator = x.as_integer_ratio()`라고 쓰고 여러 숫자 형에서 작동하도록 할 수 있습니다. (Contributed by Lisa Roach in bpo-33073 and Raymond Hettinger in bpo-37819.)
- `int`, `float` 및 `complex`의 생성자는 이제 해당 메서드 `__int__()`, `__float__()` 또는 `__complex__()`를 사용할 수 없을 때, `__index__()` 특수 메서드가 있으면 사용할 수 있습니다. (Contributed by Serhiy Storchaka in bpo-20092.)
- 정규식에 `\N{name}` 이스케이프 지원이 추가되었습니다:

```
>>> notice = 'Copyright © 2019'
>>> copyright_year_pattern = re.compile(r'\N{copyright sign}\s*(\d{4})')
>>> int(copyright_year_pattern.search(notice).group(1))
2019
```

(Contributed by Jonathan Eunice and Serhiy Storchaka in bpo-30688.)

- 이제 딕셔너리와 딕셔너리 뷰는 `reversed()`를 사용하여 삽입 순서의 역순으로 이터레이트 할 수 있습니다. (Contributed by Rémi Lapeyre in bpo-33462.)
- 함수 호출에서 키워드 이름에 허용된 문법이 더 제한되었습니다. 특히, `f((keyword)=arg)`는 더는 허용되지 않습니다. 키워드 인자 대입 항의 왼쪽에 이름 그대로 이상의 것을 허용하려는 의도는 결코 없었습니다. (Contributed by Benjamin Peterson in bpo-34641.)
- `yield`와 `return` 문에서의 일반화된 이터러블 언 패킹은 더는 둘러싸는 괄호를 요구하지 않습니다. 이는 *yield*와 *return* 문법이 일반 대입 문법과 더 잘 일치하도록 만듭니다:

```
>>> def parse(family):
        lastname, *members = family.split()
        return lastname.upper(), *members

>>> parse('simpsons homer marge bart lisa sally')
('SIMPSONS', 'homer', 'marge', 'bart', 'lisa', 'sally')
```

(Contributed by David Cuthbert and Jordan Chapman in bpo-32117.)

- `[(10, 20) (30, 40)]`와 같은 코드에서처럼 쉼표가 빠질 때, 컴파일러는 유용한 제안과 함께 `SyntaxWarning`을 표시합니다. 이것은 단지 첫 번째 튜플이 콜러블이 아니라고 알리는 `TypeError`를 개선한 것입니다. (Contributed by Serhiy Storchaka in bpo-15248.)
- `datetime.date`나 `datetime.datetime`의 서브 클래스와 `datetime.timedelta` 객체 간의 산술 연산은 이제 베이스 클래스가 아닌 서브 클래스의 인스턴스를 반환합니다. 이는 `astimezone()`과 같은 `datetime.timedelta` 산술을 직접 또는 간접적으로 사용하는 연산의 반환형에도 영향을 줍니다. (Contributed by Paul Ganssle in bpo-32417.)
- 파이썬 인터프리터가 Ctrl-C(SIGINT)에 의해 인터럽트 되고 그로 인한 `KeyboardInterrupt` 예외가 잡히지 않으면, 파이썬 프로세스는 이제 Ctrl-C로 인해 죽었다는 것을 호출한 프로세스가 감지할 수 있도록 SIGINT 시그널을 통해서나 올바른 종료 코드를 통해 종료합니다. POSIX와 윈도우의 셸은 대화식 세션에서 스크립트를 올바르게 종료하기 위해 이를 사용합니다. (Contributed by Google via Gregory P. Smith in bpo-1054041.)
- 일부 고급 프로그래밍 스타일은 기존 함수에 대한 `types.CodeType` 객체를 갱신할 필요가 있습니다. 코드 객체는 불변이므로, 기존 코드 객체에 기반하는 새 코드 객체를 만들어야 합니다. 19개의 매개 변수 때문에, 다소 지루했습니다. 이제, 새로운 `replace()` 메서드를 사용하면 몇 가지 변경된 매개 변수만으로 복제본을 만들 수 있습니다.

  다음은 *data* 매개 변수가 키워드 인자로 사용되지 않도록 `statistics.mean()` 함수를 변경하는 예입니다:

```
>>> from statistics import mean
>>> mean(data=[10, 20, 90])
40
>>> mean.__code__ = mean.__code__.replace(co_posonlyargcount=1)
>>> mean(data=[10, 20, 90])
Traceback (most recent call last):
  ...
TypeError: mean() got some positional-only arguments passed as keyword␣
↪arguments: 'data'
```

(Contributed by Victor Stinner in bpo-37032.)

- 정수의 경우, `pow()` 함수의 세 인자 형식은 이제 밑이 모듈러스와 서로소일 때 음수 지수를 허용합니다. 지수가 `-1`일 때 밑에 대한 모듈러 역수를 계산하고, 다른 음수 지수에 대해서는 그 역수의 적절한

거듭제곱을 계산합니다. 예를 들어, 모듈로 137에 대한 38의 모듈러 역수를 계산하려면, 다음과 같이 작성하십시오:

```
>>> pow(38, -1, 137)
119
>>> 119 * 38 % 137
1
```

모듈러 역수는 선형 디오판토스 방정식(linear Diophantine equations)의 해에서 나타납니다. 예를 들어, 4258□ + 147□ = 369에 대한 정수해를 찾으려면, 먼저 4258□ ≡ 369 (mod 147)로 다시 쓴 다음 해를 구하십시오:

```
>>> x = 369 * pow(4258, -1, 147) % 147
>>> y = (4258 * x - 369) // -147
>>> 4258 * x + 147 * y
369
```

(Contributed by Mark Dickinson in bpo-36027.)

- 키가 먼저 계산되고 값이 두 번째로 계산되도록, 딕셔너리 컴프리헨션은 딕셔너리 리터럴과 동기화되었습니다:

```
>>> # Dict comprehension
>>> cast = {input('role? '): input('actor? ') for i in range(2)}
role? King Arthur
actor? Chapman
role? Black Knight
actor? Cleese

>>> # Dict literal
>>> cast = {input('role? '): input('actor? ')}
role? Sir Robin
actor? Eric Idle
```

키 표현식에서 대입된 변수를 값 표현식에서 사용할 수 있기 때문에, 보장된 실행 순서는 대입 표현식에 유용합니다:

```
>>> names = ['Martin von Löwis', 'Łukasz Langa', 'Walter Dörwald']
>>> {(n := normalize('NFC', name)).casefold() : n for name in names}
{'martin von löwis': 'Martin von Löwis',
 'łukasz langa': 'Łukasz Langa',
 'walter dörwald': 'Walter Dörwald'}
```

(Contributed by Jörn Heissler in bpo-35224.)

- `object.__reduce__()` 메서드는 이제 2개에서 6개 요소 길이의 튜플을 반환할 수 있습니다. 이전에는 5가 한계였습니다. 새로운 선택적 여섯 번째 요소는 `(obj, state)` 서명을 갖는 콜러블입니다. 이를 통해 특정 객체의 상태 갱신 동작을 직접 제어할 수 있습니다. *None*이 아니면, 이 콜러블은 객체의 `__setstate__()` 메서드보다 우선합니다. (Contributed by Pierre Glaser and Olivier Grisel in bpo-35900.)

4 새 모듈

- 새로운 `importlib.metadata` 모듈은 제삼자 패키지에서 메타 데이터를 읽을 수 있도록 (잠정적으로) 지원합니다. 예를 들어, 설치된 패키지의 버전 번호, 진입점 목록 등을 추출 할 수 있습니다:

```
>>> # Note following example requires that the popular "requests"
>>> # package has been installed.
>>>
>>> from importlib.metadata import version, requires, files
>>> version('requests')
'2.22.0'
>>> list(requires('requests'))
['chardet (<3.1.0,>=3.0.2)']
>>> list(files('requests'))[:5]
[PackagePath('requests-2.22.0.dist-info/INSTALLER'),
 PackagePath('requests-2.22.0.dist-info/LICENSE'),
 PackagePath('requests-2.22.0.dist-info/METADATA'),
 PackagePath('requests-2.22.0.dist-info/RECORD'),
 PackagePath('requests-2.22.0.dist-info/WHEEL')]
```

(Contributed by Barry Warsaw and Jason R. Coombs in bpo-34632.)

5 개선된 모듈

5.1 ast

이제 AST 노드는 노드 끝의 정확한 위치를 제공하는 `end_lineno`와 `end_col_offset` 어트리뷰트를 가집니다. (이것은 `lineno`와 `col_offset` 어트리뷰트가 있는 노드에만 적용됩니다.)

새 함수 `ast.get_source_segment()`는 특정 AST 노드의 소스 코드를 반환합니다.

(Contributed by Ivan Levkivskyi in bpo-33416.)

`ast.parse()` 함수에는 몇 가지 새로운 플래그가 있습니다:

- `type_comments=True`는 특정 AST 노드와 연관된 **PEP 484**와 **PEP 526** 형 주석의 텍스트를 반환하도록 합니다;
- `mode='func_type'`은 **PEP 484** "서명 형 주석"(함수 정의 AST 노드에 대해 반환됩니다)을 구문 분석하는 데 사용될 수 있습니다;
- `feature_version=(3, N)`은 이전 버전의 파이썬 3을 지정할 수 있게 합니다. 예를 들어, `feature_version=(3, 4)`는 `async`와 `await`를 예약어가 아닌 단어로 취급합니다.

(Contributed by Guido van Rossum in bpo-35766.)

5.2 asyncio

`asyncio.run()`은 임시 API를 졸업하고 안정 API가 되었습니다. 이 함수는 이벤트 루프를 자동으로 관리하면서 코루틴을 실행하고 결과를 반환하는 데 사용할 수 있습니다. 예를 들면:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

asyncio.run(main())
```

이것은 대략 다음과 동등합니다:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

loop = asyncio.new_event_loop()
asyncio.set_event_loop(loop)
try:
    loop.run_until_complete(main())
finally:
    asyncio.set_event_loop(None)
    loop.close()
```

실제 구현은 훨씬 더 복잡합니다. 따라서, `asyncio.run()`은 asyncio 프로그램을 실행하는 데 선호되는 방법이어야 합니다.

(Contributed by Yury Selivanov in bpo-32314.)

`python -m asyncio`를 실행하면 네이티브 하게 비동기 REPL을 시작합니다. 이를 통해 최상위 `await`가 있는 코드를 빠르게 실험할 수 있습니다. 더는 호출할 때마다 새로운 이벤트 루프를 만드는 `asyncio.run()`을 직접 호출할 필요가 없습니다:

```
$ python -m asyncio
asyncio REPL 3.8.0
Use "await" directly instead of "asyncio.run()".
Type "help", "copyright", "credits" or "license" for more information.
>>> import asyncio
>>> await asyncio.sleep(10, result='hello')
hello
```

(Contributed by Yury Selivanov in bpo-37028.)

`asyncio.CancelledError` 예외는 이제 `Exception`이 아닌 `BaseException`을 상속하고, 더는 `concurrent.futures.CancelledError`를 상속하지 않습니다. (Contributed by Yury Selivanov in bpo-32528.)

윈도우에서, 기본 이벤트 루프는 이제 `ProactorEventLoop` 입니다. (Contributed by Victor Stinner in bpo-34687.)

`ProactorEventLoop`는 이제 UDP도 지원합니다. (Contributed by Adam Meily and Andrew Svetlov in bpo-29883.)

이제 `ProactorEventLoop`가 `KeyboardInterrupt`("CTRL+C")에 의해 인터럽트 될 수 있습니다. (Contributed by Vladimir Matveev in bpo-23057.)

`asyncio.Task` 내에 래핑 된 코루틴을 얻기 위한 `asyncio.Task.get_coro()`를 추가했습니다. (Contributed by Alex Grönholm in bpo-36999.)

이제 `name` 키워드 인자를 `asyncio.create_task()`나 `create_task()` 이벤트 루프 메서드에 전달하거나, 태스크 객체의 `set_name()` 메서드를 호출하여 asyncio 태스크의 이름을 지정할 수 있습니다. 태스크 이름은 `asyncio.Task`의 `repr()` 출력에 표시되며 `get_name()` 메서드를 사용하여 조회할 수도 있습니다. (Contributed by Alex Grönholm in bpo-34270.)

`asyncio.loop.create_connection()`에 Happy Eyeballs에 대한 지원이 추가되었습니다. 동작을 지정하기 위해, 두 개의 매개변수가 추가되었습니다: *happy_eyeballs_delay*와 *interleave*. Happy Eyeballs 알고리즘은 IPv4와 IPv6을 지원하는 응용 프로그램에서 둘 모두를 사용하여 동시에 연결을 시도하여 응답성을 향상합니다. (Contributed by twisteroid ambassador in bpo-33530.)

5.3 builtins

내장 `compile()`이 `ast.PyCF_ALLOW_TOP_LEVEL_AWAIT` 플래그를 받아들이도록 개선되었습니다. 이 새 플래그가 전달되면, `compile()`은 일반적으로 유효하지 문법으로 간주하는 최상위 수준 `await`, `async for` 및 `async with` 구문을 허용합니다. 그러면 `CO_COROUTINE` 플래그로 표시된 비동기 코드 객체가 반환될 수 있습니다. (Contributed by Matthias Bussonnier in bpo-34616)

5.4 collections

이제 `collections.namedtuple()`의 `_asdict()` 메서드는 `collections.OrderedDict` 대신 `dict`를 반환합니다. 파이썬 3.7부터 일반 딕셔너리가 순서를 보장하기 때문에 이것이 가능합니다. `OrderedDict`의 추가 기능이 필요하면, 제안된 해결 방법은 원하는 형으로 결과를 캐스팅하는 것입니다: `OrderedDict(nt._asdict())`. (Contributed by Raymond Hettinger in bpo-35864.)

5.5 cProfile

`cProfile.Profile` 클래스는 이제 컨텍스트 관리자로 사용할 수 있습니다. 다음처럼 코드 블록을 프로파일 하십시오:

```
import cProfile

with cProfile.Profile() as profiler:
      # code to be profiled
      ...
```

(Contributed by Scott Sanderson in bpo-29235.)

5.6 csv

`csv.DictReader`는 이제 `collections.OrderedDict` 대신 `dict`의 인스턴스를 반환합니다. 이 도구는 이제 여전히 필드 순서를 유지하면서 더 빠르고 메모리를 덜 사용합니다. (Contributed by Michael Selik in bpo-34003.)

5.7 curses

하부 ncurses 라이브러리에 대한 구조화된 버전 정보를 담는 새로운 변수를 추가했습니다: `ncurses_version`. (Contributed by Serhiy Storchaka in bpo-31680.)

5.8 ctypes

윈도우에서, `CDLL` 및 서브 클래스는 이제 하부 `LoadLibraryEx` 호출에 대한 플래그를 지정하는 *winmode* 매개 변수를 받아들입니다. 기본 플래그는 DLL이 저장된 경로(전체나 부분 경로가 초기 DLL을 로드하는 데 사용되었다면)와 `add_dll_directory()`로 추가된 경로를 포함하여 신뢰할 수 있는 위치에서만 DLL 종속성을 로드하도록 설정됩니다. (Contributed by Steve Dower in bpo-36085.)

5.9 datetime

ISO 년, 주 번호 및 요일로 `date`와 `datetime` 객체를 각각 생성하는, 새로운 대체 생성자 `datetime.date.fromisocalendar()` 와 `datetime.datetime.fromisocalendar()`를 추가했습니다; 이것들은 각 클래스의 `isocalendar` 메서드의 역입니다. (Contributed by Paul Ganssle in bpo-36004.)

5.10 functools

이제 `functools.lru_cache()`는 데코레이터를 반환하는 함수가 아닌 직접적인 데코레이터로 사용할 수 있습니다. 그래서 이 두 가지가 모두 지원됩니다:

```
@lru_cache
def f(x):
    ...

@lru_cache(maxsize=256)
def f(x):
    ...
```

(Contributed by Raymond Hettinger in bpo-36772.)

인스턴스 수명 동안 캐시 된 계산된 속성을 위한 새로운 `functools.cached_property()` 데코레이터를 추가했습니다.

```
import functools
import statistics

class Dataset:
   def __init__(self, sequence_of_numbers):
      self.data = sequence_of_numbers

   @functools.cached_property
   def variance(self):
      return statistics.variance(self.data)
```

(Contributed by Carl Meyer in bpo-21145)

싱글 디스패치를 사용하여 메서드를 제네릭 함수로 변환하는 새로운 `functools.singledispatchmethod()` 데코레이터를 추가했습니다:

```
from functools import singledispatchmethod
from contextlib import suppress

class TaskManager:

    def __init__(self, tasks):
        self.tasks = list(tasks)

    @singledispatchmethod
    def discard(self, value):
        with suppress(ValueError):
            self.tasks.remove(value)

    @discard.register(list)
    def _(self, tasks):
        targets = set(tasks)
        self.tasks = [x for x in self.tasks if x not in targets]
```

(Contributed by Ethan Smith in bpo-32380)

5.11 gc

이제 `get_objects()`는 객체를 가져올 세대(generation)를 나타내는 선택적 *generation* 매개 변수를 받을 수 있습니다. (Contributed by Pablo Galindo in bpo-36016.)

5.12 gettext

`pgettext()`와 그 변형을 추가했습니다. (Contributed by Franz Glasner, Éric Araujo, and Cheryl Sabella in bpo-2504.)

5.13 gzip

재현성 있는 출력을 위해 `gzip.compress()`에 *mtime* 매개 변수를 추가했습니다. (Contributed by Guo Ci Teo in bpo-34898.)

특정 유형의 유효하지 않거나 손상된 gzip 파일의 경우 이제 `OSError` 대신 `BadGzipFile` 예외가 발생합니다. (Contributed by Filip Gruszczyński, Michele Orrù, and Zackery Spytz in bpo-6584.)

5.14 IDLE과 idlelib

N 줄(기본적으로 50)을 초과하는 출력은 버튼으로 축소됩니다. N은 설정 대화 상자의 General 페이지의 PyShell 섹션에서 변경할 수 있습니다. 그보다 작은, 하지만 아주 길 수도 있는, 줄은 출력을 마우스 오른쪽 버튼으로 클릭하면 압착할 수 있습니다. 압착된 출력은 버튼을 더블 클릭해서 제자리에서 확장하거나, 버튼을 마우스 오른쪽 단추로 클릭하여 클립 보드나 별도의 창으로 확장할 수 있습니다. (Contributed by Tal Einat in bpo-1529353.)

사용자 정의 설정으로 모듈을 실행하기 위해 Run 메뉴에 "Run Customized"를 추가합니다. 입력한 모든 명령 줄 인자는 sys.argv에 추가됩니다. 또한 다음 사용자 정의 실행을 위해 상자에 다시 나타납니다. 일반적인 셸 메인 모듈 재시작을 억제할 수도 있습니다. (Contributed by Cheryl Sabella, Terry Jan Reedy, and others in bpo-5680 and bpo-37627.)

IDLE 편집기 창에 선택적인 줄 번호를 추가했습니다. 창은 구성 대화 상자의 General 탭에서 설정하지 않는 한 줄 번호 없이 열립니다. 기존 창의 줄 번호는 옵션 메뉴에서 표시되거나 숨겨집니다. (Contributed by Tal Einat and Saimadhav Heblikar in bpo-17535.)

OS 네이티브 인코딩은 이제 파이썬 문자열과 Tcl 객체 간 변환에 사용됩니다. 이를 통해 IDLE은 그림 이모티콘과 기타 BMP가 아닌 문자를 사용할 수 있습니다. 이러한 문자는 표시하거나 클립 보드에 복사하고 클립보드에서 붙여넣을 수 있습니다. 문자열을 Tcl에서 파이썬으로 변환했다가 되돌리는 것은 이제 실패하지 않습니다. (많은 사람이 8년 동안 이 작업을 했지만, 마침내 Serhiy Storchaka가 이 문제를 bpo-13153에서 해결했습니다.)

New in 3.8.1:

Add option to toggle cursor blink off. (Contributed by Zackery Spytz in bpo-4603.)

Escape key now closes IDLE completion windows. (Contributed by Johnny Najera in bpo-38944.)

위의 변경 사항은 3.7 유지 보수 릴리스로 역 이식되었습니다.

Add keywords to module name completion list. (Contributed by Terry J. Reedy in bpo-37765.)

5.15 inspect

`inspect.getdoc()` 함수는 이제 `__slots__` 어트리뷰트가 값이 독스트링인 `dict`면 독스트링을 찾을 수 있습니다. 이것은 `property()`, `classmethod()` 및 `staticmethod()`에 대해 이미 가지고 있는 것과 유사한 문서화 옵션을 제공합니다:

```
class AudioClip:
    __slots__ = {'bit_rate': 'expressed in kilohertz to one decimal place',
                 'duration': 'in seconds, rounded up to an integer'}
    def __init__(self, bit_rate, duration):
        self.bit_rate = round(bit_rate / 1000.0, 1)
        self.duration = ceil(duration)
```

(Contributed by Raymond Hettinger in bpo-36326.)

5.16 io

개발 모드(`-X env`)와 디버그 빌드에서, `close()` 메서드가 실패하면 `io.IOBase` 파이널라이저가 이제 예외를 로그 합니다. 릴리스 빌드에서는 기본적으로 예외가 조용히 무시됩니다. (Contributed by Victor Stinner in bpo-18748.)

5.17 itertools

`itertools.accumulate()` 함수는 초깃값을 지정하기 위한 옵션 *initial* 키워드 인자를 추가했습니다:

```
>>> from itertools import accumulate
>>> list(accumulate([10, 5, 30, 15], initial=1000))
[1000, 1010, 1015, 1045, 1060]
```

(Contributed by Lisa Roach in bpo-34659.)

5.18 json.tool

모든 입력 행을 별도의 JSON 객체로 구문 분석하는 `--json-lines` 옵션을 추가했습니다. (Contributed by Weipeng Hong in bpo-31553.)

5.19 logging

`logging.basicConfig()`에 *force* 키워드 인자를 추가했습니다. 참으로 설정하면, 루트 로거에 연결된 기존 처리기는 다른 인자로 지정된 구성을 수행하기 전에 제거되고 닫힙니다.

이것은 오랜 문제를 해결합니다. 일단 로거나 *basicConfig()*가 호출되면, *basicConfig()*에 대한 후속 호출은 조용히 무시되었습니다. 이로 인해 대화식 프롬프트나 Jupyter 노트북을 사용하여 다양한 로깅 구성 옵션을 갱신, 실험 또는 가르치기가 어려웠습니다.

(Suggested by Raymond Hettinger, implemented by Dong-hee Na, and reviewed by Vinay Sajip in bpo-33897.)

5.20 math

두 점 사이의 유클리드 거리를 계산하기 위한 새로운 함수 `math.dist()` 가 추가되었습니다. (Contributed by Raymond Hettinger in bpo-33089.)

다중 차원을 처리하도록 `math.hypot()` 함수를 확장했습니다. 이전에는, 2-D 케이스 만 지원했습니다. (Contributed by Raymond Hettinger in bpo-33089.)

'start' 값(기본값: 1)과 숫자의 이터러블의 곱을 반환하는 `sum()` 의 대응 물인 새 함수 `math.prod()` 를 추가했습니다:

```
>>> prior = 0.8
>>> likelihoods = [0.625, 0.84, 0.30]
>>> math.prod(likelihoods, start=prior)
0.126
```

(Contributed by Pablo Galindo in bpo-35606.)

새로운 조합 함수(combinatoric functions) `math.perm()` 와 `math.comb()` 가 추가되었습니다:

```
>>> math.perm(10, 3)     # Permutations of 10 things taken 3 at a time
720
>>> math.comb(10, 3)     # Combinations of 10 things taken 3 at a time
120
```

(Contributed by Yash Aggarwal, Keller Fuchs, Serhiy Storchaka, and Raymond Hettinger in bpo-37128, bpo-37178, and bpo-35431.)

정수 제곱근을 부동 소수점으로 변환하지 않고 정확하게 계산하기 위한 새로운 함수 `math.isqrt()` 를 추가했습니다. 새 함수는 제한 없이 큰 정수를 지원합니다. `floor(sqrt(n))` 보다 빠르지만 `math.sqrt()` 보다 느립니다:

```
>>> r = 650320427
>>> s = r ** 2
>>> isqrt(s - 1)          # correct
650320426
>>> floor(sqrt(s - 1))    # incorrect
650320427
```

(Contributed by Mark Dickinson in bpo-36887.)

함수 `math.factorial()` 은 더는 int-류가 아닌 인자를 받아들이지 않습니다. (Contributed by Pablo Galindo in bpo-33083.)

5.21 mmap

`mmap.mmap` 클래스에는 이제 `madvise()` 시스템 호출에 액세스하는 `madvise()` 메서드가 있습니다. (Contributed by Zackery Spytz in bpo-32941.)

5.22 multiprocessing

새로운 `multiprocessing.shared_memory` 모듈을 추가했습니다. (Contributed by Davin Potts in bpo-35813.)

macOS에서, 이제 기본적으로 *spawn* 시작 방법이 사용됩니다. (Contributed by Victor Stinner in bpo-33725.)

5.23 os

확장 모듈을 임포트 하거나 ctypes를 사용하여 DLL을 로드할 때 네이티브 종속성에 대한 추가 검색 경로를 제공하기 위해 윈도우에서 새로운 함수 add_dll_directory() 가 추가되었습니다. (Contributed by Steve Dower in bpo-36085.)

새로운 os.memfd_create() 함수가 추가되어 memfd_create() 시스템 호출을 감쌉니다. (Contributed by Zackery Spytz and Christian Heimes in bpo-26836.)

윈도우에서 재해석 지점(reparse points - 심볼릭 링크와 디렉터리 정션(directory junction)을 포함합니다)을 처리하기 위한 많은 수동 논리가 운영 체제로 위임되었습니다. 특히, os.stat() 은 이제 운영 체제에서 지원하는 모든 것을 탐색하지만, os.lstat() 은 "이름 서로게이트(name surrogates)"로 식별되는 재해석 지점만 열고 다른 이름은 os.stat() 처럼 엽니다. 모든 경우에, stat_result.st_mode는 심볼릭 링크에만 S_IFLNK를 설정하고 다른 유형의 재해석 지점에는 설정하지 않습니다. 다른 유형의 재해석 지점을 식별하려면, 새 stat_result.st_reparse_tag 어트리뷰트를 확인하십시오.

윈도우에서, os.readlink() 는 이제 디렉터리 정션을 읽을 수 있습니다. islink() 는 디렉터리 정션에 대해 False를 반환함에 유의하십시오. 따라서 islink를 먼저 확인하는 코드는 정션을 계속 디렉터리로 처리하지만, os.readlink() 의 에러를 처리하는 코드는 정션을 이제 링크로 처리할 수 있습니다.

(Contributed by Steve Dower in bpo-37834.)

5.24 os.path

exists(), lexists(), isdir(), isfile(), islink() 및 ismount() 와 같은 불리언 결과를 반환하는 os.path 함수는 이제 OS 수준에서 표현할 수 없는 문자나 바이트를 포함하는 경로에 대해 ValueError 나 그것의 서브 클래스 UnicodeEncodeError 와 UnicodeDecodeError 를 발생시키는 대신 False 를 반환합니다. (Contributed by Serhiy Storchaka in bpo-33721.).

윈도우에서 expanduser() 는 이제 USERPROFILE 환경 변수를 선호하며 일반적으로 일반 사용자 계정에 설정되지 않는 HOME을 사용하지 않습니다. (Contributed by Anthony Sottile in bpo-36264.)

윈도우에서 isdir() 은 존재하지 않는 디렉터리에 대한 링크에 대해 더는 True를 반환하지 않습니다.

윈도우에서 realpath() 는 이제 심볼릭 링크와 디렉터리 정션을 포함하는 재해석 지점을 결정(resolve)합니다.

(Contributed by Steve Dower in bpo-37834.)

5.25 pathlib

exists(), is_dir(), is_file(), is_mount(), is_symlink(), is_block_device(), is_char_device(), is_fifo(), is_socket() 과 같은 불리언 결과를 반환하는 pathlib.Path 메서드는 OS 수준에서 표현할 수 없는 문자가 포함된 경로에 대해 ValueError나 그것의 서브 클래스 UnicodeEncodeError 를 발생시키는 대신 False를 반환합니다. (Contributed by Serhiy Storchaka in bpo-33721.)

경로를 가리키는 하드 링크를 만드는 pathlib.Path.link_to() 를 추가했습니다. (Contributed by Joannah Nanjekye in bpo-26978)

5.26 pickle

C 최적화된 `Pickler`를 서브 클래싱하는 `pickle` 확장은 이제 특수 `reducer_override()` 메서드를 정의하여 함수와 클래스의 피클링 로직을 재정의할 수 있습니다. (Contributed by Pierre Glaser and Olivier Grisel in bpo-35900.)

5.27 plistlib

새로운 `plistlib.UID`가 추가되었으며 NSKeyedArchiver 인코딩된 바이너리 plist를 읽고 쓰는 지원을 활성화했습니다. (Contributed by Jon Janzen in bpo-26707.)

5.28 pprint

`pprint` 모듈은 여러 함수에 *sort_dicts* 매개 변수를 추가했습니다. 기본적으로, 이러한 함수는 렌더링이나 인쇄 전에 딕셔너리를 계속 정렬합니다. 그러나, *sort_dicts*가 거짓으로 설정되면, 딕셔너리는 키가 삽입된 순서를 유지합니다. 디버깅하는 동안 JSON 입력과 비교하는 데 유용 할 수 있습니다.

또한, `pprint.pprint()`와 유사하지만 *sort_dicts*가 `False`로 기본 설정된 새로운 편의 함수 `pprint.pp()`가 있습니다:

```
>>> from pprint import pprint, pp
>>> d = dict(source='input.txt', operation='filter', destination='output.txt')
>>> pp(d, width=40)                  # Original order
{'source': 'input.txt',
 'operation': 'filter',
 'destination': 'output.txt'}
>>> pprint(d, width=40)              # Keys sorted alphabetically
{'destination': 'output.txt',
 'operation': 'filter',
 'source': 'input.txt'}
```

(Contributed by Rémi Lapeyre in bpo-30670.)

5.29 py_compile

`py_compile.compile()`은 이제 조용한 모드를 지원합니다. (Contributed by Joannah Nanjekye in bpo-22640.)

5.30 shlex

새 `shlex.join()` 함수는 `shlex.split()`의 역함수로 작동합니다. (Contributed by Bo Bayles in bpo-32102.)

5.31 shutil

`shutil.copytree()`는 이제 새로운 `dirs_exist_ok` 키워드 인자를 받아들입니다. (Contributed by Josh Bronson in bpo-20849.)

`shutil.make_archive()`는 이제 이식성과 표준 적합성을 향상하기 위해 새로운 아카이브에 최신 pax (POSIX.1-2001) 형식을 기본으로 합니다, `tarfile` 모듈에 대한 해당 변경 사항을 상속했습니다. (Contributed by C.A.M. Gerlach in bpo-30661.)

윈도우에서 `shutil.rmtree()`는 이제 그 내용을 먼저 재귀적으로 삭제하지 않고 디렉터리 정션을 삭제합니다. (Contributed by Steve Dower in bpo-37834.)

5.32 socket

같은 소켓에서 IPv4와 IPv6 연결을 모두 받아들이는 것을 포함하여 일반적으로 서버 소켓을 만들 때 필요한 작업을 자동화하는 `create_server()`와 `has_dualstack_ipv6()` 편리 함수가 추가되었습니다. (Contributed by Giampaolo Rodolà in bpo-17561.)

`socket.if_nameindex()`, `socket.if_nametoindex()` 및 `socket.if_indextoname()` 함수가 윈도우에서 구현되었습니다. (Contributed by Zackery Spytz in bpo-37007.)

5.33 ssl

TLS 1.3 포스트 핸드 셰이크 인증을 활성화하는 `post_handshake_auth`와 시작하는 `verify_client_post_handshake()`를 추가했습니다. (Contributed by Christian Heimes in bpo-34670.)

5.34 statistics

`statistics.mean()`의 더 빠른 부동 소수점 변형으로 `statistics.fmean()`을 추가했습니다. (Contributed by Raymond Hettinger and Steven D'Aprano in bpo-35904.)

`statistics.geometric_mean()`을 추가했습니다 (Contributed by Raymond Hettinger in bpo-27181.)

가장 빈번한 값의 리스트를 반환하는 `statistics.multimode()`가 추가되었습니다. (Contributed by Raymond Hettinger in bpo-35892.)

데이터나 분포를 등분위 간격(equiprobable intervals)(예를 들어, 4분위 수(quattiles), 십분위 수(deciles) 또는 백분위 수(percentiles))으로 나누는 `statistics.quantiles()`를 추가했습니다. (Contributed by Raymond Hettinger in bpo-36546.)

무작위 변수의 정규 분포를 만들고 조작하기 위한 도구인 `statistics.NormalDist`를 추가했습니다. (Contributed by Raymond Hettinger in bpo-36018.)

```
>>> temperature_feb = NormalDist.from_samples([4, 12, -3, 2, 7, 14])
>>> temperature_feb.mean
6.0
>>> temperature_feb.stdev
6.356099432828281

>>> temperature_feb.cdf(3)           # Chance of being under 3 degrees
0.3184678262814532
>>> # Relative chance of being 7 degrees versus 10 degrees
>>> temperature_feb.pdf(7) / temperature_feb.pdf(10)
1.2039930378537762

>>> el_niño = NormalDist(4, 2.5)
>>> temperature_feb += el_niño       # Add in a climate effect
>>> temperature_feb
NormalDist(mu=10.0, sigma=6.830080526611674)

>>> temperature_feb * (9/5) + 32     # Convert to Fahrenheit
NormalDist(mu=50.0, sigma=12.294144947901014)
>>> temperature_feb.samples(3)       # Generate random samples
[7.672102882379219, 12.000027119750287, 4.647488369766392]
```

5.35 sys

"발생시킬 수 없는 예외(unraisable exceptions)" 처리 방법을 제어하기 위해 재정의할 수 있는 새로운 `sys.unraisablehook()` 함수를 추가했습니다. 예외가 발생했지만, 파이썬에서 처리할 방법이 없을 때 호출됩니다. 예를 들어, 파괴자(destructor)가 예외를 발생시키거나 가비지 수집(`gc.collect()`) 중에. (Contributed by Victor Stinner in bpo-36829.)

5.36 tarfile

`tarfile` 모듈은 새로운 아카이브를 만들 때 이제 이전의 GNU 전용이 아닌 최신 pax (POSIX.1-2001) 형식을 기본값으로 사용합니다. 이것은 표준화되고 확장 가능한 형식으로 일관된 인코딩(UTF-8)을 사용하여 플랫폼 간 이식성을 개선하고, 몇 가지 다른 이점을 제공합니다. (Contributed by C.A.M. Gerlach in bpo-36268.)

5.37 threading

잡히지 않은 `threading.Thread.run()` 예외를 처리하는 새로운 `threading.excepthook()` 함수를 추가했습니다. 잡히지 않은 `threading.Thread.run()` 예외가 처리되는 방식을 제어하기 위해 재정의될 수 있습니다. (Contributed by Victor Stinner in bpo-1230540.)

새로운 `threading.get_native_id()` 함수와 `native_id` 어트리뷰트를 `threading.Thread` 클래스에 추가합니다. 이것들은 커널이 할당한 현재 스레드의 네이티브 정수 스레드 ID를 반환합니다. 이 기능은 특정 플랫폼에서만 사용할 수 있습니다, 자세한 내용은 `get_native_id`를 참조하십시오. (Contributed by Jake Tesler in bpo-36084.)

5.38 tokenize

`tokenize` 모듈은 이제 후행 줄 넘김이 없는 입력을 제공할 때 묵시적으로 `NEWLINE` 토큰을 출력합니다. 이 동작은 이제 C 토크나이저가 내부적으로 수행하는 것과 일치합니다. (Contributed by Ammar Askar in bpo-33899.)

5.39 tkinter

`tkinter.Spinbox` 클래스에 `selection_from()`, `selection_present()`, `selection_range()` 및 `selection_to()` 메서드를 추가했습니다. (Contributed by Juliette Monsel in bpo-34829.)

`tkinter.Canvas` 클래스에 `moveto()` 메서드를 추가했습니다. (Contributed by Juliette Monsel in bpo-23831.)

`tkinter.PhotoImage` 클래스는 이제 `transparency_get()`과 `transparency_set()` 메서드를 갖습니다. (Contributed by Zackery Spytz in bpo-25451.)

5.40 time

macOS 10.12를 위한 새로운 시계 `CLOCK_UPTIME_RAW`가 추가되었습니다. (Contributed by Joannah Nanjekye in bpo-35702.)

5.41 typing

typing 모듈은 몇 가지 새로운 기능을 통합했습니다:

- 키별 형이 있는 딕셔너리 형. **PEP 589**와 typing.TypedDict를 참조하십시오. TypedDict는 문자열 키만 사용합니다. 기본적으로, 모든 키가 있어야 합니다. 키를 선택적으로 만들려면 "total=False"를 지정하십시오:

```
class Location(TypedDict, total=False):
    lat_long: tuple
    grid_square: str
    xy_coordinate: tuple
```

- 리터럴 형. **PEP 586**과 typing.Literal을 참조하십시오. Literal 형은 매개 변수나 반환 값이 하나 이상의 특정 리터럴 값으로 제한됨을 나타냅니다:

```
def get_status(port: int) -> Literal['connected', 'disconnected']:
    ...
```

- "최종(final)" 변수, 함수, 메서드 및 클래스. **PEP 591**, typing.Final 및 typing.final()을 참조하십시오. final 한정자는 정적 형 검사기에 서브 클래싱, 재정의 또는 재대입을 제한하도록 지시합니다:

```
pi: Final[float] = 3.1415926536
```

- 프로토콜 정의. **PEP 544**, typing.Protocol 및 typing.runtime_checkable()을 참조하십시오. typing.SupportsInt와 같은 간단한 ABC는 이제 Protocol 서브 클래스입니다.
- 새 프로토콜 클래스 typing.SupportsIndex.
- 새 함수 typing.get_origin()과 typing.get_args().

5.42 unicodedata

unicodedata 모듈은 유니코드 12.1.0 릴리스를 사용하도록 업그레이드되었습니다.

새 함수 is_normalized()를 문자열이 특정 정규화 형식인지 확인하는 데 사용할 수 있습니다. 종종 실제로 문자열을 정규화하는 것보다 훨씬 빠릅니다. (Contributed by Max Belanger, David Euresti, and Greg Price in bpo-32285 and bpo-37966).

5.43 unittest

Mock의 비동기 버전을 지원하는 AsyncMock이 추가되었습니다. 테스트를 위해 적절한 새로운 assert 함수들이 추가되었습니다. (Contributed by Lisa Roach in bpo-26467).

setUpModule()과 setUpClass()에 대한 정리를 지원하기 위해 unittest에 addModuleCleanup() 과 addClassCleanup()를 추가했습니다. (Contributed by Lisa Roach in bpo-24412.)

여러 mock assert 함수는 이제 실패 시 실제 호출의 리스트도 인쇄합니다. (Contributed by Petter Strandmark in bpo-35047.)

unittest 모듈은 unittest.IsolatedAsyncioTestCase를 통해 코루틴을 테스트 케이스로 사용하는 지원을 얻었습니다. (Contributed by Andrew Svetlov in bpo-32972.)

예:

```
import unittest


class TestRequest(unittest.IsolatedAsyncioTestCase):

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    async def asyncSetUp(self):
        self.connection = await AsyncConnection()

    async def test_get(self):
        response = await self.connection.get("https://example.com")
        self.assertEqual(response.status_code, 200)

    async def asyncTearDown(self):
        await self.connection.close()


if __name__ == "__main__":
    unittest.main()
```

5.44 venv

venv는 이제 PowerShell Core 6.1에서 가상 환경을 활성화하기 위한 모든 플랫폼에 Activate.ps1 스크립트를 포함합니다. (Contributed by Brett Cannon in bpo-32718.)

5.45 weakref

weakref.proxy()에 의해 반환된 프락시 객체는 이제 다른 숫자 연산자에 더해 행렬 곱셈 연산자 @와 @=을 지원합니다. (Contributed by Mark Dickinson in bpo-36669.)

5.46 xml

DTD 및 외부 엔티티 조회에 대한 완화로서, xml.dom.minidom과 xml.sax 모듈은 기본적으로 더는 외부 엔티티를 처리하지 않습니다. (Contributed by Christian Heimes in bpo-17239.)

xml.etree.ElementTree 모듈의 .find*() 메서드는 이름 공간을 무시하는 {*}tag와 지정된 이름 공간의 모든 태그를 반환하는 {namespace}*와 같은 와일드카드 검색을 지원합니다. (Contributed by Stefan Behnel in bpo-28238.)

xml.etree.ElementTree 모듈은 C14N 2.0을 구현하는 새로운 함수 -xml.etree.ElementTree.canonicalize()를 제공합니다. (Contributed by Stefan Behnel in bpo-13611.)

xml.etree.ElementTree.XMLParser의 대상 객체는 새 콜백 메서드 start_ns()와 end_ns()를 통해 이름 공간 선언 이벤트를 수신할 수 있습니다. 또한, xml.etree.ElementTree.TreeBuilder 대상은 주석과 처리 명령어에 대한 이벤트를 처리하여 생성된 트리에 포함하도록 구성할 수 있습니다. (Contributed by Stefan Behnel in bpo-36676 and bpo-36673.)

5.47 xmlrpc

xmlrpc.client.ServerProxy는 이제 각 요청과 함께 보낼 HTTP 헤더의 시퀀스를 위한 선택적 *headers* 키워드 인자를 지원합니다. 무엇보다도, 기본 베이식 인증(basic authentication)에서 더 빠른 세션 인증(session authentication)으로 업그레이드 할 수 있도록 합니다. (Contributed by Cédric Krier in bpo-35153.)

6 최적화

- subprocess 모듈은 이제 성능 향상을 위해 몇몇 경우 os.posix_spawn() 함수를 사용할 수 있습니다. 현재, 다음과 같은 조건이 모두 충족될 때 macOS와 리눅스(glibc 2.24 이상 사용)에서만 사용됩니다:
  - *close_fds*가 거짓입니다.
  - *preexec_fn*, *pass_fds*, *cwd* 및 *start_new_session* 매개 변수가 설정되지 않았습니다.
  - *executable* 경로가 디렉터리를 포함합니다.

  (Contributed by Joannah Nanjekye and Victor Stinner in bpo-35537.)

- shutil.copyfile(), shutil.copy(), shutil.copy2(), shutil.copytree() 및 shutil.move()는 파일을 보다 효율적으로 복사하기 위해 리눅스와 macOS에서 플랫폼별 "빠른 복사(fast-copy)" 시스템 호출을 사용합니다. "빠른 복사"는 복사 작업이 커널 내에서 발생하여 "outfd.write(infd.read())"처럼 파이썬에서 사용자 공간(userspace) 버퍼를 사용하지 않도록 합니다. 윈도우에서 shutil.copyfile()은 더 큰 기본 버퍼 크기(16 KiB 대신 1 MiB)를 사용하고 shutil.copyfileobj()의 memoryview() 기반 변형이 사용됩니다. 같은 파티션 내에서 512 MiB 파일을 복사할 때 속도 개선은 리눅스에서는 약 +26%, macOS에서는 +50%, 윈도우에서는 +40%입니다. 또한, 훨씬 적은 CPU 사이클이 소모됩니다. shutil-platform-dependent-efficient-copy-operations 절을 참조하십시오. (Contributed by Giampaolo Rodolà in bpo-33671.)

- shutil.copytree()는 os.scandir() 함수를 사용하고 이것에 의존하는 모든 복사 함수는 캐시된 os.stat() 값을 사용합니다. 8000개의 파일이 있는 디렉터리를 복사할 때 속도 개선은 리눅스에서는 +9%, 윈도우에서는 +20%, 윈도우 SMB 공유에서는 +30%입니다. 또한, os.stat() 시스템 호출 수가 38% 감소하여 특히 shutil.copytree()가 네트워크 파일 시스템에서 더 빠릅니다. (Contributed by Giampaolo Rodolà in bpo-33695.)

- pickle 모듈의 기본 프로토콜은 이제 파이썬 3.4에서 처음 소개된 프로토콜 4입니다. 파이썬 3.0부터 사용 가능한 프로토콜 3보다 성능이 우수하고 크기가 작습니다.

- PyGC_Head에서 Py_ssize_t 멤버 하나를 삭제했습니다. 모든 GC 추적 객체(예를 들어, 튜플, 리스트, 딕셔너리)의 크기는 4 또는 8바이트 줄어듭니다. (Contributed by Inada Naoki in bpo-33597.)

- uuid.UUID는 이제 __slots__를 사용하여 메모리 사용량을 줄입니다. (Contributed by Wouter Bolsterlee and Tal Einat in bpo-30977)

- operator.itemgetter()의 33% 성능 향상. 인자 처리를 최적화하고 튜플에 대한 하나의 음수가 아닌 정수 인덱스의 흔한 경우에 대한 빠른 경로를 추가했습니다(표준 라이브러리의 일반적인 사용 사례입니다). (Contributed by Raymond Hettinger in bpo-35664.)

- collections.namedtuple()에서의 필드 조회 속도 향상. 이제 두 배 이상 빨라져, 파이썬에서 가장 빠른 인스턴스 변수 조회 형식이 되었습니다. (Contributed by Raymond Hettinger, Pablo Galindo, and Joe Jevnik, Serhiy Storchaka in bpo-32492.)

- list 생성자는 입력 이터러블의 길이를 알 수 있으면 (입력이 __len__을 구현하면) 내부 항목 버퍼를 초과 할당하지 않습니다. 이것은 생성된 리스트가 평균 12% 작게 만듭니다. (Contributed by Raymond Hettinger and Pablo Galindo in bpo-33234.)

- 클래스 변수 쓰기 속도 두 배 향상. 특수하지 않은(non-dunder) 어트리뷰트가 갱신되었을 때, 슬롯 갱신을 위한 불필요한 호출이 있었습니다. (Contributed by Stefan Behnel, Pablo Galindo Salgado, Raymond Hettinger, Neil Schemenauer, and Serhiy Storchaka in bpo-36012.)

- 많은 내장 함수와 메서드에 전달된 인자를 변환하는 오버헤드 감소. 이것은 몇몇 간단한 내장 함수와 메서드 호출을 20–50%까지 가속합니다. (Contributed by Serhiy Storchaka in bpo-23867, bpo-35582 and bpo-36127.)

- LOAD_GLOBAL 명령어는 이제 새로운 "옵코드(opcode) 단위 캐시" 메커니즘을 사용합니다. 이제 약 40% 빨라졌습니다. (Contributed by Yury Selivanov and Inada Naoki in bpo-26219.)

7 빌드와 C API 변경

- 기본 `sys.abiflags`는 빈 문자열이 되었습니다: pymalloc을 위한 `m` 플래그가 쓸모없게 되었으므로 (pymalloc을 포함하거나 포함하지 않는 빌드는 ABI 호환입니다) 제거되었습니다. (Contributed by Victor Stinner in bpo-36707.)

  변경의 예:

  - `python3.8` 프로그램만 설치되고, `python3.8m` 프로그램은 사라졌습니다.
  - `python3.8-config` 스크립트만 설치되고, `python3.8m-config` 스크립트는 사라졌습니다.
  - `m` 플래그는 동적 라이브러리 파일명의 접미사에서 제거되었습니다: 표준 라이브러리는 물론 PyPI에서 다운로드 한 것과 같은 제삼자 패키지에서 생성되고 설치된 확장 모듈. 예를 들어, 리눅스에서 파이썬 3.7 접미사 `.cpython-37m-x86_64-linux-gnu.so`는 파이썬 3.8에서 `.cpython-38-x86_64-linux-gnu.so`가 되었습니다.

- 여러 종류의 API를 더 잘 구분할 수 있도록 헤더 파일이 재구성되었습니다:

  - `Include/*.h`는 이식성 있는 공용 안정 C API 여야 합니다.
  - `Include/cpython/*.h`는 CPython에 고유한 불안정한 C API 여야 합니다; `_Py`나 `_PY`로 시작하는 일부 비공개 API가 포함된 공개 API.
  - `Include/internal/*.h`는 아주 CPython에 특정한 비공개 내부 C API입니다. 이 API는 과거 호환성 보증이 없어서, CPython 외부에서 사용해서는 안 됩니다. 이것은 함수를 호출하지 않고 CPython 내부에 액세스해야 하는 디버거나 프로파일과 같이 매우 구체적인 요구 사항을 위해서만 노출됩니다. 이 API는 이제 `make install`에 의해 설치됩니다.

  (Contributed by Victor Stinner in bpo-35134 and bpo-35081, Eric Snow가 파이썬 3.7에서 시작한 작업.)

- 일부 매크로는 정적 인라인 함수로 변환되었습니다: 매개 변수 형과 반환형이 잘 정의되며, 매크로에 특정한 문제가 없으며, 변수는 지역 스코프를 갖습니다. 예:

  - `Py_INCREF()`, `Py_DECREF()`
  - `Py_XINCREF()`, `Py_XDECREF()`
  - `PyObject_INIT()`, `PyObject_INIT_VAR()`
  - 비공개 함수: `_PyObject_GC_TRACK()`, `_PyObject_GC_UNTRACK()`, `_Py_Dealloc()`

  (Contributed by Victor Stinner in bpo-35059.)

- `PyByteArray_Init()`와 `PyByteArray_Fini()` 함수가 제거되었습니다. 파이썬 2.7.4와 파이썬 3.2.0부터 아무것도 하지 않고, 제한된 API(안정 ABI)에서 제외되었으며 문서로 만들어지지 않았습니다. (Contributed by Victor Stinner in bpo-35713.)

- `PyExceptionClass_Name()`의 결과는 이제 `char *` 대신 `const char *` 형입니다. (Contributed by Serhiy Storchaka in bpo-33818.)

- `Modules/Setup.dist`와 `Modules/Setup`의 이중성이 제거되었습니다. 이전에는, CPython 소스 트리를 갱신할 때, 업스트림 변경 사항을 반영하기 위해 `Modules/Setup.dist`(소스 트리 내부)를 `Modules/Setup`(빌드 트리 내부)으로 수동으로 복사해야 했습니다. 이는 패키지 작성자에게는 작은 이점을 제공했지만, 파일 복사를 잊어버리면 빌드가 실패할 수 있어서 CPython 개발자들을 자주 귀찮게 만들었습니다.

  이제 빌드 시스템은 항상 소스 트리 안의 `Modules/Setup`에서 읽습니다. 그 파일을 사용자 정의하고 싶은 사람들은 소스 트리에 대한 다른 변경과 마찬가지로 CPython의 git 포크나 패치 파일로 변경 사항을 유지하는 것이 좋습니다.

  (Contributed by Antoine Pitrou in bpo-32430.)

- `PyLong_AsLong()`과 같은 파이썬 숫자를 C 정수로 변환하는 함수와, `PyArg_ParseTuple()` 처럼 `'i'`와 같은 정수 변환 포맷 단위로 인자를 구문 분석하는 함수는, 이제 사용할 수 있다면 `__int__()` 대신 `__index__()` 특수 메서드를 사용합니다. `__int__()` 메서드가 있지만 `__index__()` 메서드가 없는 객체(`Decimal`과 `Fraction` 같은)에 대해서 폐지 경고가 발생

합니다. PyNumber_Check()는 이제 __index__()를 구현하는 객체에 대해 1을 반환합니다. PyNumber_Long(), PyNumber_Float() 및 PyFloat_AsDouble()은 이제 사용할 수 있으면 __index__() 메서드를 사용합니다. (Contributed by Serhiy Storchaka in bpo-36048 and bpo-20092.)

- 힙에 할당된 형 객체는 이제 PyType_GenericAlloc()이 아닌 PyObject_Init()(그리고 이것과 평행한 매크로 PyObject_INIT)에서 참조 횟수를 증가시킵니다. 인스턴스 할당이나 할당 해제를 수정하는 형은 조정이 필요할 수 있습니다. (Contributed by Eddie Elizondo in bpo-35810.)
- 새 함수 PyCode_NewWithPosOnlyArgs()는 PyCode_New()처럼 코드 객체를 만들 수 있지만, 위치 전용 인자의 개수를 나타내는 *posonlyargcount* 매개 변수가 추가로 있습니다. (Contributed by Pablo Galindo in bpo-37221.)
- Py_SetPath()는 이제 sys.executable을 프로그램 이름(Py_GetProgramName()) 대신 프로그램 전체 경로(Py_GetProgramFullPath())로 설정합니다. (Contributed by Victor Stinner in bpo-38234.)

8 폐지

- distutils bdist_wininst 명령은 이제 폐지되었습니다, 대신 bdist_wheel(wheel 패키지)을 사용하십시오. (Contributed by Victor Stinner in bpo-37481.)
- ElementTree 모듈의 폐지된 메서드 getchildren()과 getiterator()는 이제 PendingDeprecationWarning 대신 DeprecationWarning을 발생시킵니다. 이것들은 파이썬 3.9에서 제거될 것입니다. (Contributed by Serhiy Storchaka in bpo-29209.)
- concurrent.futures.ThreadPoolExecutor의 인스턴스가 아닌 객체를 loop.set_default_executor()로 전달하는 것은 폐지되었고 파이썬 3.9에서 금지될 것입니다. (Contributed by Elvis Pranskevichus in bpo-34075.)
- xml.dom.pulldom.DOMEventStream, wsgiref.util.FileWrapper 및 fileinput.FileInput의 __getitem__() 메서드는 폐지되었습니다.

  이 메서드의 구현은 *index* 매개 변수를 무시하고 대신 다음 항목을 반환했습니다. (Contributed by Berker Peksag in bpo-9372.)
- typing.NamedTuple 클래스는 _field_types 어트리뷰트를 폐지했고, 같은 정보를 갖는 __annotations__ 어트리뷰트로 대신합니다. (Contributed by Raymond Hettinger in bpo-36320.)
- ast 클래스 Num, Str, Bytes, NameConstant 및 Ellipsis는 폐지된 것으로 간주하고 향후 파이썬 버전에서 제거될 예정입니다. 대신 Constant를 사용해야 합니다. (Contributed by Serhiy Storchaka in bpo-32892.)
- ast.NodeVisitor 메서드 visit_Num(), visit_Str(), visit_Bytes(), visit_NameConstant() 및 visit_Ellipsis()는 이제 폐지되었으며 향후 파이썬 버전에서는 호출되지 않을 것입니다. 모든 상수 노드를 처리하기 위해서는 visit_Constant() 메서드를 추가하십시오. (Contributed by Serhiy Storchaka in bpo-36917.)
- asyncio.coroutine() 데코레이터는 폐지되었고 버전 3.10에서 제거됩니다. @asyncio.coroutine 대신, async def를 사용하십시오. (Contributed by Andrew Svetlov in bpo-36921.)
- asyncio에서, 다음에 대해 *loop* 인자의 명시적 전달은 폐지되었고 버전 3.10에서 제거됩니다: asyncio.sleep(), asyncio.gather(), asyncio.shield(), asyncio.wait_for(), asyncio.wait(), asyncio.as_completed(), asyncio.Task, asyncio.Lock, asyncio.Event, asyncio.Condition, asyncio.Semaphore, asyncio.BoundedSemaphore, asyncio.Queue, asyncio.create_subprocess_exec() 및 asyncio.create_subprocess_shell().
- 코루틴 객체를 asyncio.wait()에 명시적으로 전달하는 것은 폐지되었고 버전 3.11에서 제거됩니다. (Contributed by Yury Selivanov in bpo-34790.)
- gettext 모듈에서는 다음 함수와 메서드가 폐지되었습니다: lgettext(), ldgettext(), lngettext() 및 ldngettext(). 이들은 인코딩된 바이트열을 반환하며, 번역된 문자열에 인

코딩 문제가 있을 때 예기치 않은 유니코드 관련 예외가 발생할 수 있습니다. 파이썬 3에서 유니코드 문자열을 반환하는 대안을 사용하는 것이 훨씬 낫습니다. 이 함수들은 오랫동안 망가져 있었습니다.

함수 `bind_textdomain_codeset()`, 메서드 `output_charset()` 과 `set_output_charset()`, 함수 `translation()`과 `install()`의 *codeset* 매개 변수도 폐지되었습니다, `l*gettext()` 함수에서만 사용되기 때문입니다. (Contributed by Serhiy Storchaka in bpo-33710.)

- `threading.Thread`의 `isAlive()` 메서드가 폐지되었습니다. (Contributed by Dong-hee Na in bpo-35283.)
- 정수 인자를 받아들이는 많은 내장과 확장 함수는 이제 `Decimal`들, `Fraction`들 및 정수로 변환될 때 손실을 수반하는 다른 객체들(예를 들어, `__int__()` 메서드는 있지만 `__index__()` 메서드는 없는 것들)에 대해 폐지 경고를 발생시킵니다. 이후 버전에서는 에러가 될 것입니다. (Contributed by Serhiy Storchaka in bpo-36048.)
- 다음 인자들을 키워드 인자로 전달하는 것이 폐지되었습니다:
  - `functools.partialmethod()`, `weakref.finalize()`, `profile.Profile.runcall()`, `cProfile.Profile.runcall()`, `bdb.Bdb.runcall()`, `trace.Trace.runfunc()` 및 `curses.wrapper()`의 *func*.
  - `unittest.TestCase.addCleanup()`의 *function*.
  - `concurrent.futures.ThreadPoolExecutor` 과 `concurrent.futures.ProcessPoolExecutor`의 `submit()` 메서드의 *fn*.
  - `contextlib.ExitStack.callback()`, `contextlib.AsyncExitStack.callback()` 및 `contextlib.AsyncExitStack.push_async_callback()`의 *callback*
  - `multiprocessing.managers.Server` 와 `multiprocessing.managers.SharedMemoryServer`의 `create()` 메서드의 *c*와 *typeid*.
  - `weakref.finalize()`의 *obj*.

  파이썬의 향후 배포에서, 이것들은 위치 전용이 될 것입니다. (Contributed by Serhiy Storchaka in bpo-36492.)

9 API 및 기능 제거

다음 기능과 API는 파이썬 3.8에서 제거되었습니다.:

- 파이썬 3.3부터, `collections`에서 ABC를 임포트 하는 것은 폐지되었고, `collections.abc`에서 임포트를 수행해야 합니다. collections에서 임포트할 수 있는 것은 3.8에서 제거될 것으로 표시되었지만 3.9로 지연되었습니다. (bpo-36952 참조)
- 파이썬 3.7에서 폐지된, `macpath` 모듈이 제거되었습니다. (Contributed by Victor Stinner in bpo-35471.)
- 파이썬 3.3부터 폐지된 이후로, `platform.popen()` 함수는 제거되었습니다: 대신 `os.popen()`을 사용하십시오. (Contributed by Victor Stinner in bpo-35345.)
- 파이썬 3.3부터 폐지된 이후로, `time.clock()` 함수는 제거되었습니다: 잘 정의된 동작을 위해, 요구 사항에 따라 `time.perf_counter()`나 `time.process_time()`을 대신 사용하십시오. (Contributed by Matthias Bussonnier in bpo-36895.)
- `pyvenv` 스크립트를 제거하고 `python3.8 -m venv`로 대체하여, 어떤 파이썬 인터프리터가 `pyvenv` 스크립트와 연결되어 있는지에 대한 혼란을 없애줍니다. (Contributed by Brett Cannon in bpo-25427.)
- `cgi` 모듈에서 `parse_qs`, `parse_qsl` 및 `escape`가 제거되었습니다. 파이썬 3.2 이하에서 폐지되었습니다. 대신 `urllib.parse`와 `html` 모듈에서 임포트 해야 합니다.
- `filemode` 함수가 `tarfile` 모듈에서 제거되었습니다. 문서로 만들어지지 않았고, 파이썬 3.3부터 폐지되었습니다.

- `XMLParser` 생성자는 더는 *html* 인자를 받아들이지 않습니다. 효과가 있었던 적이 없고 3.4에서 폐지되었습니다. 다른 모든 매개 변수는 이제 키워드 전용입니다. (Contributed by Serhiy Storchaka in bpo-29209.)
- `XMLParser`의 `doctype()` 메서드를 제거했습니다. (Contributed by Serhiy Storchaka in bpo-29209.)
- "unicode_internal" 코덱이 제거되었습니다. (Contributed by Inada Naoki in bpo-36297.)
- `sqlite3` 모듈의 `Cache`와 `Statement` 객체는 사용자에게 노출되지 않습니다. (Contributed by Aviv Palivoda in bpo-30262.)
- `fileinput.input()`과 `fileinput.FileInput()`의 `bufsize` 키워드 인자는 무시 되었고 파이썬 3.6부터 폐지되었으며, 이제 제거되었습니다. bpo-36952 (Contributed by Matthias Bussonnier.)
- 파이썬 3.7에서 폐지된 `sys.set_coroutine_wrapper()`와 `sys.get_coroutine_wrapper()` 함수는 제거되었습니다; bpo-36933 (Contributed by Matthias Bussonnier.)

10 파이썬 3.8로 이식하기

이 절에서는 여러분의 코드 수정을 요구할 수도 있는 이전에 설명한 변경 사항과 다른 버그 수정 사항을 나열합니다.

10.1 파이썬 동작의 변경

- 일드 표현식(`yield`와 `yield from` 절 모두)은 이제 컴프리헨션과 제너레이터 표현식에서 허용되지 않습니다 (가장 왼쪽 `for` 절의 이터러블 표현식 제외). (Contributed by Serhiy Storchaka in bpo-10544.)
- 아이덴티티 검사(`is`와 `is not`)가 특정 형의 리터럴(예를 들어, 문자열, 숫자)과 함께 사용될 때 컴파일러는 이제 `SyntaxWarning`을 생성합니다. 이들은 종종 CPython에서 우연히 작동할 수 있지만, 언어 명세에 의해 보장되지는 않습니다. 이 경고는 사용자가 동등 검사(==와 !=)를 대신 사용하도록 권장합니다. (Contributed by Serhiy Storchaka in bpo-34850.)
- CPython 인터프리터는 때에 따라 예외를 삼킬 수 있습니다. 파이썬 3.8에서는 이런 경우가 덜 발생합니다. 특히, 형 딕셔너리에서 어트리뷰트를 가져올 때 발생하는 예외는 더는 무시되지 않습니다. (Contributed by Serhiy Storchaka in bpo-35459.)
- 내장형 `bool`, `int`, `float`, `complex` 및 표준 라이브러리의 일부 클래스에서 `__str__` 구현을 제거했습니다. 이제 이들은 `object`에서 `__str__()`을 상속합니다. 결과적으로, 이러한 클래스의 서브 클래스에서 `__repr__()` 메서드를 정의하면 문자열 표현에 영향을 줍니다. (Contributed by Serhiy Storchaka in bpo-36793.)
- AIX에서, `sys.platform`은 더는 주(major) 버전을 포함하지 않습니다. `'aix3'` .. `'aix7'` 대신 항상 `'aix'`입니다. 이전 버전의 파이썬에서는 버전 번호가 포함되어 있기 때문에, 항상 `sys.platform.startswith('aix')`를 사용하는 것이 좋습니다. (Contributed by M. Felt in bpo-36588.)
- `PyEval_AcquireLock()`과 `PyEval_AcquireThread()`는 인터프리터가 파이널라이즈하는 동안 호출되면 현재 스레드를 종료하여, `PyEval_RestoreThread()`, `Py_END_ALLOW_THREADS()` 및 `PyGILState_Ensure()`와 일관되게 만듭니다. 이 동작을 원하지 않으면, `_Py_IsFinalizing()`이나 `sys.is_finalizing()`을 확인하여 호출을 보호하십시오. (Contributed by Joannah Nanjekye in bpo-36475.)

10.2 파이썬 API의 변경

- `os.getcwdb()` 함수는 이제 윈도우에서 ANSI 코드 페이지가 아닌 UTF-8 인코딩을 사용합니다. 이유는 **PEP 529**를 참조하십시오. 이 함수는 윈도우에서 더는 폐지되지 않았습니다. (Contributed by Victor Stinner in bpo-37412.)
- `subprocess.Popen`은 이제 어떤 경우에 더 나은 성능을 위해 `os.posix_spawn()`을 사용할 수 있습니다. 리눅스용 윈도우 하위 시스템과 QEMU 사용자 에뮬레이션에서, `os.posix_spawn()`을 사용하는 `Popen` 생성자는 더는 "프로그램 누락"과 같은 에러에 예외를 발생시키지 않습니다. 대신 자식 프로세스는 0이 아닌 `returncode`로 실패합니다. (Contributed by Joannah Nanjekye and Victor Stinner in bpo-35537.)
- `subprocess.Popen`의 *preexec_fn* 인자는 더는 서브 인터프리터와 호환되지 않습니다. 서브 인터프리터에서 매개 변수를 사용하면 `RuntimeError`가 발생합니다. (Contributed by Eric Snow in bpo-34651, modified by Christian Heimes in bpo-37951.)
- `imap.IMAP4.logout()` 메서드는 더는 조용히 임의의 예외를 무시하지 않습니다. (Contributed by Victor Stinner in bpo-36348.)
- 파이썬 3.3부터 폐지된 이후로, `platform.popen()` 함수는 제거되었습니다: 대신 `os.popen()`을 사용하십시오. (Contributed by Victor Stinner in bpo-35345.)
- `statistics.mode()` 함수는 다중 모드(multimodal) 데이터가 주어질 때 더는 예외를 발생시키지 않습니다. 대신, 입력 데이터에서 만나는 첫 번째 모드를 반환합니다. (Contributed by Raymond Hettinger in bpo-35892.)
- `tkinter.ttk.Treeview` 클래스의 `selection()` 메서드는 더는 인자를 받아들이지 않습니다. 선택을 변경하기 위해 인자와 함께 사용하는 것은 파이썬 3.6에서 폐지되었습니다. 선택을 변경하려면 `selection_set()`과 같은 특화된 메서드를 사용하십시오. (Contributed by Serhiy Storchaka in bpo-31508.)
- `xml.dom.minidom` 모듈의 `writexml()`, `toxml()` 및 `toprettyxml()` 메서드와 `xml.etree`의 `write()` 메서드는 이제 사용자가 지정한 어트리뷰트 순서를 보존합니다. (Contributed by Diego Rojas and Raymond Hettinger in bpo-34160.)
- 플래그 `'r'`로 열린 `dbm.dumb` 데이터베이스는 이제 읽기 전용입니다. `'r'`과 `'w'` 플래그로 `dbm.dumb.open()`하면 더는 데이터베이스가 없을 때 만들지 않습니다. (Contributed by Serhiy Storchaka in bpo-32749.)
- `XMLParser`의 서브 클래스에 정의된 `doctype()` 메서드는 더는 호출되지 않으며 `DeprecationWarning` 대신 `RuntimeWarning`을 방출합니다. XML doctype 선언을 처리하려면 대상에 `doctype()` 메서드를 정의하십시오. (Contributed by Serhiy Storchaka in bpo-29209.)
- 사용자 지정 메타 클래스가 `type.__new__`에 전달된 이름 공간에 `__classcell__` 항목을 제공하지 않으면 이제 `RuntimeError`가 발생합니다. 파이썬 3.6–3.7에서는 `DeprecationWarning`이 방출되었습니다. (Contributed by Serhiy Storchaka in bpo-23722.)
- `cProfile.Profile` 클래스는 이제 컨텍스트 관리자로 사용할 수 있습니다. (Contributed by Scott Sanderson in bpo-29235.)
- `shutil.copyfile()`, `shutil.copy()`, `shutil.copy2()`, `shutil.copytree()` 및 `shutil.move()`는 플랫폼 특정 "고속 복사(fast-copy)" 시스템 호출을 사용합니다 (shutil-platform-dependent-efficient-copy-operations 절을 참조하십시오).
- 윈도우에서 `shutil.copyfile()` 기본 버퍼 크기가 16 KiB에서 1 MiB로 변경되었습니다.
- `PyGC_Head` 구조체가 완전히 변경되었습니다. 구조체 멤버를 건드리는 모든 코드는 다시 작성해야 합니다. (bpo-33597 참조.)
- `PyInterpreterState` 구조체는 "내부" 헤더 파일(구체적으로 Include/internal/pycore_pystate.h)로 이동했습니다. 불투명한 `PyInterpreterState`는 공용 API(와 안정 ABI)의 일부로 계속 사용할 수 있습니다. 설명서는 구조체의 필드가 공용이 아니라고 말하고 있으니, 아무도 해당 필드를 사용하지 않기를 바랍니다. 그러나, 하나 이상의 내부 필드에 의지하고 대안이 없으면 BPO 이슈를 여십시오. 조정을 돕기 위해 노력할 것입니다 (공용 API에 접근 함수를 추가하는 것도 가능합니다). (bpo-35886을 참조하십시오.)

- `mmap.flush()` 메서드는 이제 모든 플랫폼에서 성공 시 `None`을 반환하고 에러 시 예외를 발생시킵니다. 이전에는, 그 동작이 플랫폼에 따라 달랐습니다: 윈도우에서 성공 시 0이 아닌 값이 반환되었습니다; 에러 시 0이 반환되었습니다. 유닉스에서 성공 시 0 값이 반환되었습니다; 에러 시 예외가 발생했습니다. (Contributed by Berker Peksag in bpo-2122.)
- `xml.dom.minidom`과 `xml.sax` 모듈은 기본적으로 더는 외부 엔티티를 처리하지 않습니다. (Contributed by Christian Heimes in bpo-17239.)
- 읽기 전용 dbm 데이터베이스(`dbm.dumb`, `dbm.gnu` 또는 `dbm.ndbm`)에서 키를 삭제하면 `KeyError` 대신 `error`(`dbm.dumb.error`, `dbm.gnu.error` 또는 `dbm.ndbm.error`)를 발생시킵니다. (Contributed by Xiang Zhang in bpo-33106.)
- 리터럴을 위해 단순화된 AST. 모든 상수는 `ast.Constant` 인스턴스로 표시됩니다. 이전 클래스 `Num`, `Str`, `Bytes`, `NameConstant` 및 `Ellipsis`를 인스턴스 화하면 `Constant` 인스턴스가 반환됩니다. (Contributed by Serhiy Storchaka in bpo-32892.)
- 윈도우에서 `expanduser()`는 이제 `USERPROFILE` 환경 변수를 선호하며 일반적으로 일반 사용자 계정에 설정되지 않는 `HOME`을 사용하지 않습니다. (Contributed by Anthony Sottile in bpo-36264.)
- `asyncio.CancelledError` 예외는 이제 `Exception`이 아닌 `BaseException`을 상속하고, 더는 `concurrent.futures.CancelledError`를 상속하지 않습니다. (Contributed by Yury Selivanov in bpo-32528.)
- `asyncio.wait_for()` 함수는 이제 `asyncio.Task` 인스턴스를 사용할 때 취소를 올바르게 기다립니다. 이전에는, *timeout*에 도달하면 취소되어 즉시 반환되었습니다. (Contributed by Elvis Pranskevichus in bpo-32751.)
- `asyncio.BaseTransport.get_extra_info()` 함수는 이제 'socket'이 *name* 매개 변수로 전달될 때 안전하게 사용할 수 있는 소켓 객체를 반환합니다. (Contributed by Yury Selivanov in bpo-37027.)
- `asyncio.BufferedProtocol`는 졸업해서 안정 API가 되었습니다.
- 윈도우에서 `ctypes`로 로드된 확장 모듈과 DLL에 대한 DLL 종속성이 이제 더 안전하게 처리됩니다. 시스템 경로, DLL이나 PYD 파일이 들어 있는 디렉터리 및 `add_dll_directory()`로 추가된 디렉터리만 로드 시간 종속성을 위해 검색합니다. 특히, `PATH`와 현재 작업 디렉터리는 더는 사용되지 않으며, 이러한 것들에 대한 수정은 더는 일반 DLL 결정(resolution)에 영향을 주지 않습니다. 여러분의 응용 프로그램이 이러한 메커니즘에 의존한다면, `add_dll_directory()`가 있는지 확인하고, 있다면 이를 사용하여 라이브러리를 로드하는 동안 DLL 디렉터리를 추가하십시오. 윈도우 7 사용자는 윈도우 업데이트 KB2533623이 설치되었는지 확인해야 합니다(이는 설치 프로그램에서도 확인합니다). (Contributed by Steve Dower in bpo-36085.)
- pgen과 관련된 헤더 파일과 함수는 순수 파이썬 구현으로 대체 된 후에 제거되었습니다. (Contributed by Pablo Galindo in bpo-36623.)
- `types.CodeType`은 **PEP 570**에 정의된 위치 전용 인자를 지원하기 위해 생성자의 두 번째 위치에 새 매개 변수(*posonlyargcount*)를 갖습니다. 첫 번째 인자(*argcount*)는 이제 위치 인자의 총수(위치 전용 인자 포함)를 나타냅니다. `types.CodeType`의 새로운 `replace()` 메서드를 사용하면 코드가 미래에도 안전하게 만들 수 있습니다.
- The parameter `digestmod` for `hmac.new()` no longer uses the MD5 digest by default.

10.3 C API의 변경

- `PyCompilerFlags` 구조체에 새로운 *cf_feature_version* 필드가 생겼습니다. `PY_MINOR_VERSION`으로 초기화해야합니다. 이 필드는 기본적으로 무시되며, `PyCF_ONLY_AST` 플래그가 *cf_flags*에 설정되었을 때, 또 그때만 사용됩니다. (Contributed by Guido van Rossum in bpo-35766.)
- `PyEval_ReInitThreads()` 함수가 C API에서 제거되었습니다. 명시적으로 호출하면 안 됩니다: 대신 `PyOS_AfterFork_Child()`를 사용하십시오. (Contributed by Victor Stinner in bpo-36728.)
- 유닉스에서, C 확장은 더는 안드로이드와 Cygwin을 제외하고는 libpython에 링크되지 않습니다. 파이썬이 내장될 때, `libpython`은 `RTLD_LOCAL`이 아니라 `RTLD_GLOBAL`로 로드되어야 합니다. 이전에는, `RTLD_LOCAL`을 사용하여, `Modules/Setup`의 `*shared*` 섹션에 의해 빌드된 표준 라이브

러리의 C 확장과 같이 libpython에 링크되지 않은 C 확장을 로드할 수 없었습니다. (Contributed by Victor Stinner in bpo-21536.)

- PY_SSIZE_T_CLEAN이 정의되지 않은 상태에서 값을 구문 분석하거나 빌드할 때(예를 들어, PyArg_ParseTuple(), Py_BuildValue(), PyObject_CallFunction() 등) 포맷의 # 변형을 사용하면 DeprecationWarning이 발생합니다. 3.10이나 4.0에서 제거됩니다. 자세한 내용은 arg-parsing를 참조하십시오. (Contributed by Inada Naoki in bpo-36381.)

- 힙에 할당된 형(가령 PyType_FromSpec()으로 만들어진 것들)의 인스턴스는 해당 형 객체에 대한 참조를 보유합니다. 이러한 형 객체의 참조 횟수를 늘리는 것이 PyType_GenericAlloc()에서 더 저수준 함수 PyObject_Init()와 PyObject_INIT()로 옮겨졌습니다. 이렇게 하면 PyType_FromSpec()을 통해 만들어진 형이, 관리되는 코드의 다른 클래스처럼 작동합니다.

  정적으로 할당된 형은 영향을 받지 않습니다.

  대다수의 경우, 부작용이 없어야 합니다. 그러나, 인스턴스를 할당한 후 수동으로 참조 횟수를 늘리는 형(아마도 버그를 회피하기 위해)이 이제 불멸이 될 수 있습니다. 이를 피하고자, 이런 클래스는 인스턴스 할당 해제 중에 형 객체에 Py_DECREF를 호출할 필요가 있습니다.

  이러한 형들을 3.8로 올바르게 이식하려면, 다음 변경 사항을 적용하십시오:

  - 인스턴스를 할당한 후 형 객체에 대한 Py_INCREF를 제거하십시오 - 있다면. 이것은 PyObject_New(), PyObject_NewVar(), PyObject_GC_New(), PyObject_GC_NewVar() 또는 PyObject_Init()나 PyObject_INIT()를 사용하는 다른 사용자 정의 할당자를 호출한 후에 발생할 수 있습니다.

    예:

    ```
    static foo_struct *
    foo_new(PyObject *type) {
        foo_struct *foo = PyObject_GC_New(foo_struct, (PyTypeObject *) type);
        if (foo == NULL)
            return NULL;
    #if PY_VERSION_HEX < 0x03080000
        // Workaround for Python issue 35810; no longer necessary in Python 3.8
        PY_INCREF(type)
    #endif
        return foo;
    }
    ```

  - 힙에 할당되는 형의 모든 사용자 정의 tp_dealloc 함수가 형의 참조 횟수를 감소시키도록 하십시오.

    예:

    ```
    static void
    foo_dealloc(foo_struct *instance) {
        PyObject *type = Py_TYPE(instance);
        PyObject_GC_Del(instance);
    #if PY_VERSION_HEX >= 0x03080000
        // This was not needed before Python 3.8 (Python issue 35810)
        Py_DECREF(type);
    #endif
    }
    ```

  (Contributed by Eddie Elizondo in bpo-35810.)

- Py_DEPRECATED() 매크로는 MSVC 용으로 구현되었습니다. 매크로는 이제 기호 이름 앞에 놓여야 합니다.

  예:

  ```
  Py_DEPRECATED(3.8) PyAPI_FUNC(int) Py_OldFunction(void);
  ```

  (Contributed by Zackery Spytz in bpo-33407.)

- 인터프리터는 더는 기능 배포에서 확장형의 바이너리 호환성을 지원하지 않습니다. 제삼자 확장 모듈에서 내보낸 PyTypeObject는 tp_finalize(Py_TPFLAGS_HAVE_FINALIZE는 tp_finalize를 읽기 전에 더는 확인되지 않습니다)를 포함하여 현재 파이썬 버전이 기대하는 모든 슬롯을 가지고 있다고 가정합니다.

  (Contributed by Antoine Pitrou in bpo-32388.)

- 함수 PyNode_AddChild()와 PyParser_AddToken()는 이제 두 개의 추가 int 인자 *end_lineno*와 *end_col_offset*을 받아들입니다.

- MinGW 도구가 python38.dll에 대해 직접 링크할 수 있도록 허용하는 libpython38.a 파일은 더는 일반 윈도우 배포에 포함되지 않습니다. 이 파일이 필요하면, MinGW binutils 패키지의 일부인 gendef와 dlltool 도구로 생성할 수 있습니다:

  ```
  gendef - python38.dll > tmp.def
  dlltool --dllname python38.dll --def tmp.def --output-lib libpython38.a
  ```

  설치된 pythonXY.dll의 위치는 윈도우의 설치 옵션과 버전 및 언어에 따라 다릅니다. 자세한 내용은 using-on-windows를 참조하십시오. 결과 라이브러리는 pythonXY.lib(일반적으로 파이썬 설치 환경의 libs 디렉터리입니다)와 같은 디렉터리에 위치해야 합니다.

  (Contributed by Steve Dower in bpo-37351.)

10.4 CPython 바이트 코드 변경

- 인터프리터 루프는 블록 스택을 언 롤링하는 로직을 컴파일러로 이동시킴으로써 단순화되었습니다. 컴파일러는 이제 값 스택을 조정하고 break, continue 및 return에 대한 정리 코드를 호출하는 명시적인 명령을 출력합니다.

  옵코드 BREAK_LOOP, CONTINUE_LOOP, SETUP_LOOP 및 SETUP_EXCEPT를 제거했습니다. 새로운 옵코드 ROT_FOUR, BEGIN_FINALLY, CALL_FINALLY 및 POP_FINALLY가 추가되었습니다. END_FINALLY와 WITH_CLEANUP_START의 동작이 변경되었습니다.

  (Contributed by Mark Shannon, Antoine Pitrou and Serhiy Storchaka in bpo-17611.)

- async for 루프에서 다음 항목을 await 할 때 발생하는 예외를 처리하기 위한 새로운 옵코드 END_ASYNC_FOR가 추가되었습니다. (Contributed by Serhiy Storchaka in bpo-33041.)

- MAP_ADD는 이제 값을 스택의 첫 번째 요소로, 키를 두 번째 요소로 기대합니다. 이 변경은 **PEP 572**의 제안에 따라, 딕셔너리 컴프리헨션에서 키가 값보다 항상 먼저 평가되도록 하기 위해 만들어졌습니다. (Contributed by Jörn Heissler in bpo-35224.)

10.5 데모와 도구

변수를 액세스하는 다양한 방법의 시간을 측정하기 위한 벤치마크 스크립트를 추가했습니다: Tools/scripts/var_access_benchmark.py. (Contributed by Raymond Hettinger in bpo-35884.)

다음은 파이썬 3.3 이후의 성능 향상에 대한 요약입니다:

Python version	3.3	3.4	3.5	3.6	3.7	3.8
Variable and attribute read access:						
read_local	4.0	7.1	7.1	5.4	5.1	3.9
read_nonlocal	5.3	7.1	8.1	5.8	5.4	4.4
read_global	13.3	15.5	19.0	14.3	13.6	7.6
read_builtin	20.0	21.1	21.6	18.5	19.0	7.5
read_classvar_from_class	20.5	25.6	26.5	20.7	19.5	18.4
read_classvar_from_instance	18.5	22.8	23.5	18.8	17.1	16.4
read_instancevar	26.8	32.4	33.1	28.0	26.3	25.4
read_instancevar_slots	23.7	27.8	31.3	20.8	20.8	20.2

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    read_namedtuple                 68.5     73.8     57.5     45.0     46.8     18.4
    read_boundmethod                29.8     37.6     37.9     29.6     26.9     27.7

Variable and attribute write access:
    write_local                      4.6      8.7      9.3      5.5      5.3      4.3
    write_nonlocal                   7.3     10.5     11.1      5.6      5.5      4.7
    write_global                    15.9     19.7     21.2     18.0     18.0     15.8
    write_classvar                  81.9     92.9     96.0    104.6    102.1     39.2
    write_instancevar               36.4     44.6     45.8     40.0     38.9     35.5
    write_instancevar_slots         28.7     35.6     36.1     27.3     26.6     25.7

Data structure read access:
    read_list                       19.2     24.2     24.5     20.8     20.8     19.0
    read_deque                      19.9     24.7     25.5     20.2     20.6     19.8
    read_dict                       19.7     24.3     25.7     22.3     23.0     21.0
    read_strdict                    17.9     22.6     24.3     19.5     21.2     18.9

Data structure write access:
    write_list                      21.2     27.1     28.5     22.5     21.6     20.0
    write_deque                     23.8     28.7     30.1     22.7     21.8     23.5
    write_dict                      25.9     31.4     33.3     29.3     29.2     24.7
    write_strdict                   22.9     28.4     29.9     27.5     25.2     23.1

Stack (or queue) operations:
    list_append_pop                144.2     93.4    112.7     75.4     74.2     50.8
    deque_append_pop                30.4     43.5     57.0     49.4     49.2     42.5
    deque_append_popleft            30.8     43.7     57.3     49.7     49.7     42.8

Timing loop:
    loop_overhead                    0.3      0.5      0.6      0.4      0.3      0.3
```

벤치마크는 python.org에서 찾을 수 있는 macOS 64비트 빌드를 실행하는 Intel® Core™ i7-4960HQ processor에서 측정되었습니다. 벤치마크 스크립트는 타이밍을 나노초로 표시합니다.

11 파이썬 3.8.1의 주목할만한 변경 사항

심각한 보안 문제로 인해, `asyncio.loop.create_datagram_endpoint()`의 *reuse_address* 매개 변수는 더는 지원되지 않습니다. 이것은 UDP에서 소켓 옵션 `SO_REUSEADDR`의 동작 때문입니다. 자세한 내용은 `loop.create_datagram_endpoint()` 설명서를 참조하십시오. (Contributed by Kyle Stanley, Antoine Pitrou, and Yury Selivanov in bpo-37228.)

12 Notable changes in Python 3.8.2

Fixed a regression with the `ignore` callback of `shutil.copytree()`. The argument types are now str and List[str] again. (Contributed by Manuel Barkhau and Giampaolo Rodola in bpo-39390.)

13 Notable changes in Python 3.8.3

The constant values of future flags in the `__future__` module are updated in order to prevent collision with compiler flags. Previously `PyCF_ALLOW_TOP_LEVEL_AWAIT` was clashing with `CO_FUTURE_DIVISION`. (Contributed by Batuhan Taskaya in bpo-39562)

14 Notable changes in Python 3.8.8

Earlier Python versions allowed using both `;` and `&` as query parameter separators in `urllib.parse.parse_qs()` and `urllib.parse.parse_qsl()`. Due to security concerns, and to conform with newer W3C recommendations, this has been changed to allow only a single separator key, with `&` as the default. This change also affects `cgi.parse()` and `cgi.parse_multipart()` as they use the affected functions internally. For more details, please see their respective documentation. (Contributed by Adam Goldschmidt, Senthil Kumaran and Ken Jin in bpo-42967.)

15 Notable changes in Python 3.8.9

A security fix alters the `ftplib.FTP` behavior to not trust the IPv4 address sent from the remote server when setting up a passive data channel. We reuse the ftp server IP address instead. For unusual code requiring the old behavior, set a `trust_server_pasv_ipv4_address` attribute on your FTP instance to `True`. (See bpo-43285)

16 Notable changes in Python 3.8.10

16.1 macOS 11.0 (Big Sur) and Apple Silicon Mac support

As of 3.8.10, Python now supports building and running on macOS 11 (Big Sur) and on Apple Silicon Macs (based on the `ARM64` architecture). A new universal build variant, `universal2`, is now available to natively support both `ARM64` and `Intel 64` in one set of executables. Note that support for "weaklinking", building binaries targeted for newer versions of macOS that will also run correctly on older versions by testing at runtime for missing features, is not included in this backport from Python 3.9; to support a range of macOS versions, continue to target for and build on the oldest version in the range.

(Originally contributed by Ronald Oussoren and Lawrence D'Anna in bpo-41100, with fixes by FX Coudert and Eli Rykoff, and backported to 3.8 by Maxime Bélanger and Ned Deily)

17 Notable changes in Python 3.8.10

17.1 urllib.parse

The presence of newline or tab characters in parts of a URL allows for some forms of attacks. Following the WHATWG specification that updates **RFC 3986**, ASCII newline `\n`, `\r` and tab `\t` characters are stripped from the URL by the parser in `urllib.parse` preventing such attacks. The removal characters are controlled by a new module level variable `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE`. (See bpo-43882)

18 Notable changes in Python 3.8.12

18.1 파이썬 API의 변경

Starting with Python 3.8.12 the `ipaddress` module no longer accepts any leading zeros in IPv4 address strings. Leading zeros are ambiguous and interpreted as octal notation by some libraries. For example the legacy function `socket.inet_aton()` treats leading zeros as octal notation. glibc implementation of modern `inet_pton()` does not accept any leading zeros.

(Originally contributed by Christian Heimes in bpo-36384, and backported to 3.8 by Achraf Merzouki)

19 Notable security feature in 3.8.14

Converting between `int` and `str` in bases other than 2 (binary), 4, 8 (octal), 16 (hexadecimal), or 32 such as base 10 (decimal) now raises a `ValueError` if the number of digits in string form is above a limit to avoid potential denial of service attacks due to the algorithmic complexity. This is a mitigation for CVE-2020-10735. This limit can be configured or disabled by environment variable, command line flag, or `sys` APIs. See the integer string conversion length limitation documentation. The default limit is 4300 digits in string form.

20 Notable Changes in 3.8.17

20.1 tarfile

- The extraction methods in `tarfile`, and `shutil.unpack_archive()`, have a new a *filter* argument that allows limiting tar features than may be surprising or dangerous, such as creating files outside the destination directory. See tarfile-extraction-filter for details. In Python 3.12, use without the *filter* argument will show a `DeprecationWarning`. In Python 3.14, the default will switch to `'data'`. (Contributed by Petr Viktorin in **PEP 706**.)

색인

H

P

R

U

Y