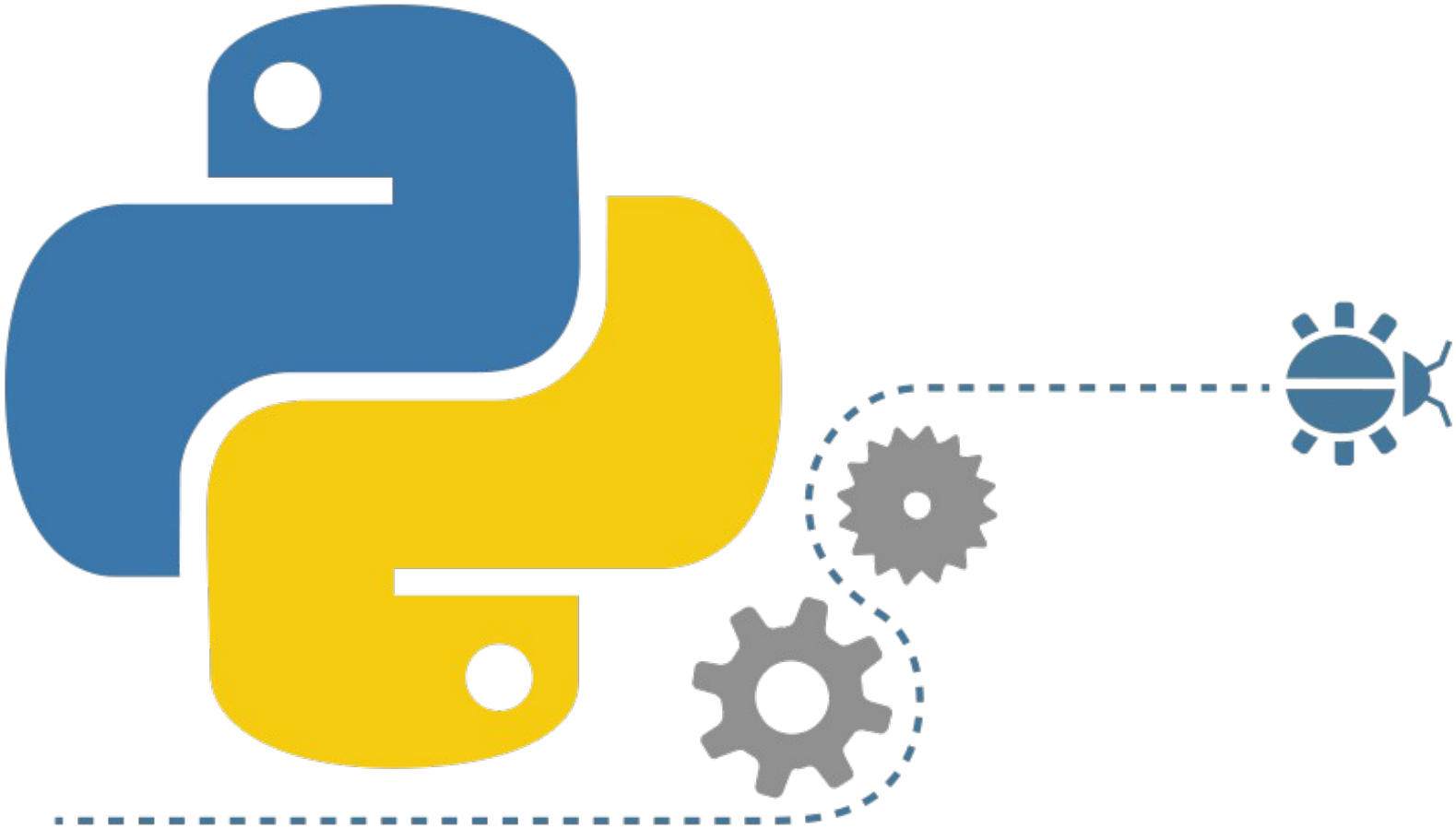

Generadores

```
 1 def generate_fibs():
 2     a, b = 0, 1
 3     while True:
 4         a, b = b, a + b
 5     yield a
 6
 7 next(g) # => 1
 8 next(g) # => 1
 9 next(g) # => 2
10 next(g) # => 3
11 next(g) # => 5
12
13
```

Programación funcional

```
1 def apply_func(func, values):
2     func(*values)
3 
4 def add_two_elements(a, b):
5     print(a + b)
6 
7 apply_func(print, [1, 2]) # 1 2
8 
9 apply_func(add_two_elements,[2, 4]) # 6
```

Programación funcional

```
 1 def make_divisibility_test(n):
 2     def divisible_by_n(m):
 3         return m % n == 0
 4     return divisible_by_n
 5
 6 is_divisable_by_five = make_divisibility_test(5)
 7 is_divisable_by_five(10) # => True
 8
 9 make_divisibility_test(7)(10) # => False
10
11 div_by_3 = make_divisibility_test(3)
12 print(filter(div_by_3, range(10))) # 0, 3, 6, 9
```

Decoradores

```
 1 def debug(func):
 2     def wrapper(*args, **kwargs):
 3         print('received arguments:', args, kwargs)
 4         return_values = func(*args, **kwargs)
 5         print('return value:', return_values)
 6         return return_values
 7     return wrapper
 8 
 9 def add_two_elements(a, b):
10     return a + b
11 
12 debug_add_two_elements = debug(add_two_elements)
13 debug_add_two_elements(5, b=6)
14 
15 @debug
16 def suma(*args):
17     return sum(args)
```

Bases de datos

- `connect(args)` – a constructor for `Connection` objects, through which access is made available. Arguments are database-dependent. So assuming `conn = connect(args)` yields a `Connection` object, we should be able to manipulate our connection via the following methods:
- `conn.close()` – close connection. This should happen by default when __del__() is called.
- `conn.commit()` – commit pending transaction (if supported).
- `conn.rollback()` – if supported by db, roll back to start of pending transaction.
- `conn.cursor()` – return a `Cursor` object for the connection. Cursors are how we manage the context of a fetch operation.

Bases de datos

- So `c = conn.cursor()` should yield a `Cursor` object. We can have multiple cursors per connection, but they are not isolated from one another. The following methods should be available:
- `c.execute[many](`*`op, [params]`*`)` – prepare and execute an operation with parameters where the second argument may be a list of parameter sequences.
- `c.fetch[one|many|all]`*`([s])`* – fetch next row, next *s* rows, or all remaining rows of result set.
- `c.close()` – close cursor.
- `c.rowcount` – number of rows produced by last execute method (-1 by default).

Bases de datos

```
import sqlite3
conn = sqlite3.connect("ejemplo.db")
c = conn.cursor()
c.execute("SELECT * FROM Clientes;")
print(c.fetchone())
conn.close()
```

Bases de datos

```
query = ("select * from sqlite_master”
         “where type='table';")
c = conn.execute(query)
res = c.fetchall()
print([res[1] for row in res])
```

Web frameworks

- Web frameworks are collections of packages or modules which allow developers to write web applications with minimal attention paid to low-level details like protocols, sockets and process management.
- Common operations implemented by web frameworks:
  - URL routing
  - Output format templating.
  - Database manipulation
  - Basic security

Web frameworks

- Django
  - Follows MVC pattern.
  - Most popular.
  - Steeper learning curve.
  - More features built-in.
- Flask
  - “Micro”-framework: minimal approach.
  - You can get things up and going much faster.
  - Less built-in functionality.
  - Also a popular option.

Flask

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
    return "hello world"
if __name__ == "__main__":
    app.run()
```

Flask

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route("/")
def hello():
    return render_template('hello.html')
if __name__ == "__main__":
    app.run()
```

Flask + db

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'sqlite:///ejemplo.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] =
False
db = SQLAlchemy(app)
db.Model.metadata.reflect(db.engine)
```

Flask + db

```
class Clientes(db.Model):
    __table__ = db.Model.metadata.tables['Clientes']
    def __repr__(self):
        return self.Nombre


Clients.query.all()
```

Flask + db

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), index=True, unique=True)
    email = db.Column(db.String(120), index=True, unique=True)
    def __repr__(self):
        return '<User {}>'.format(self.username)


db.create_all()
db.session.commit()
```

Flask + db

```
u = User(username='susan',
         email='susan@example.com')

db.session.add(u)

db.session.commit()

u = User.query.get(1)
```