FAST API CHEATSHEET

(Author: Abhi Balijepalli)

BASIC REQUIREMENTS:

>> pip install fastapi[all] uvicorn[standard]

```
from fastapi import FastAPI
app = FastAPI()
app
```

GET: To fetch something

item_id: is a path parameter for a route

```
@app.get('/items/{item_id}')
 async def read_item(item_id: int):
   return {'item_id': item_id}
```

Fast API also lets you put classes in routes using Enum

POST: To send a message

Pydantic lets you send a request body as JSON

```
from pydantic import BaseModel

class Person(BaseModel):
    name: str
    userID: int
    Occupation: Optional[str] = None

@app.post("/register/")
async def create_person(person: Person):
    return person
```

PATCH: You can use patch to partially update a document.

Just like a POST request, you need to use `Pydantic` with the parameter `exclude_unset` to receive partial updates.
You can read more about it here

DELETE: Deleting a document in a database

The following example finds a user in a DB and deletes them using a helper function

```
@app.delete("/users/{userID}", status_code=204)
def delete_user(userID: int) -> None:
   user_to_remove = find_user(userID)

   if user_to_remove is not None:
      users.remove(user_to_remove)

def find_user(userID) -> Optional[Book]:
   for user in users:
      if user.userID == userID:
         return user
   return None
```

PUT: Fetch and replace data

The put operation can fetch data and replace it with something. More examples can be found here: FAST API PUT

```
class Person(BaseModel):
    name: str
    Occupation: Optional[str] = None

@app.put("/register/{userID}")
async def update_user(userID: int, person: Person):
    return {"userID" : userID, **person.dict()}
```

RESOURCES:

Fast API Tutorial Python Types Build API from Scratch