**Richtlijnen die ChatGPT moet volgen bij het opbouwen van nieuwe Python-projecten**

**1. Hoe ziet een nieuw project eruit?**

**Projectassisten2\_0** is een hulpprogramma waarmee je nieuwe projecten kunt aanmaken.  
Wanneer je met dit hulpmiddel een project genereert, wordt er automatisch een basisstructuur klaargezet:

* **Bestanden en mappen**
  + main.py met logging (Start/Stop).
  + core/logging\_setup.py en tools/log\_report.py.
  + .projassist.json met projectinfo, paden en logging.
  + logs/ map (leeg, wordt gevuld bij run).
  + .git/ map (versiebeheer geïnitialiseerd).
  + .vscode/ map met settings.json en tasks.json.
  + eventueel extra’s: gui/MainWindow.ui, project\_summary/scripts/, enz.
* **Virtuele omgeving**
  + Basispackages geïnstalleerd (minimaal PyQt6).
* **Logging**
  + Tekstlog (logs/log.txt) en JSON-log (logs/log.jsonl).
* **Markers**
  + Alle scripts volgen de vaste marker-structuur.

👉 Zo ontstaat een nieuw project dat klaar is om verder uit te bouwen, volledig los van het hulpprogramma Projectassisten2\_0 zelf.

**2. Afspraken bij wijzigingen (uit ALLforms.txt)**

* **Altijd werken via formulieren**: ADD / REPLACE / DELETE / CREATE.
* **Markers verplicht bij code**:
  + Secties (# [SECTION: …] / # [END: SECTION: …])
  + Functies (# [FUNC: …] / # [END: FUNC: …])
  + Classes (# [CLASS: …] / # [END: CLASS: …])
  + Main-guard (# [SECTION: MAIN] / # [END: SECTION: MAIN])
* Markers **nooit** binnen bodies.
* **Logging verplicht**:
  + Nieuwe scripts → module-logger, start/stop in main/entry, 2–3 debugregels.
  + Wijzigingen → behoud logger, voeg max. enkele logregels toe.
* **Defaults**:
  + Contextregels = 3
  + JSON-registratie = ja (groep scripts)
  + Git = ja (commit & push)
  + Encoding = utf-8
  + CREATE → Bestaat-al = mislukken tenzij anders gevraagd

**Aanvulling: Hoe de Codewijziger werkt**

De **codewijziger\_controller.py** verwerkt de formulieren en ondersteunt ADD, REPLACE, DELETE en CREATE.

**Hoofdflow:**

1. Formulier laden & parsen → leest velden en controleert markers.
2. Doelbestand & markers zoeken → blok tussen Marker-van en Marker-tot.
3. Verschillen berekenen → diff/hunks, opties voor whitespace/case.
4. Wijzigingen toepassen → selectief of volledig blok; “Lock markers” mogelijk.
5. Dry-run preview → toont eindresultaat voor opslaan.
6. Opslaan → maakt .bak backup, schrijft bestand weg, registreert in .projassist.json, commit & push via Git, kan openen in editor.
7. Herstel → backup terugzetten.

**Per actie:**

* **ADD**: blok toevoegen (bij EXTENSION\_POINTS of onderaan).
* **REPLACE**: blok vervangen tussen markers.
* **DELETE**: blok verwijderen.
* **CREATE**: nieuw bestand maken met voorstel-inhoud.

**3. Markerbeleid (uit Markerbeleid.docx)**

* **Structuur**
  + Markers altijd boven/onder blokken.
  + BEGIN en END met exact dezelfde naam.
  + Sectienamen in HOOFDLETTERS.
* **Formaat**
  + IMPORTS, CONSTANTS, CLASSES, FUNCS, MAIN.
* **Toepassing**
  + Altijd: imports, constanten, classes, functies, main.
  + Nooit: binnen bodies, docstrings, XML/.ui-bestanden.
* **Consistentie**
  + Sluit END altijd af met type + naam.
  + Gebruik overal dezelfde stijl (CAPS).

**Voorbeeld:**

# [SECTION: IMPORTS]

import logging

# [END: SECTION: IMPORTS]

# [FUNC: main]

def main() -> int:

logger = logging.getLogger(\_\_name\_\_)

logger.info("Start")

return 0

# [END: FUNC: main]

# [SECTION: MAIN]

if \_\_name\_\_ == "\_\_main\_\_":

raise SystemExit(main())

# [END: SECTION: MAIN]

**4. Richtlijnen logging bij verdere opbouw (uit Logging-docx)**

**Uitgangspunt**

* main.py initialiseert logging.
* Logs → logs/log.txt en logs/log.jsonl.
* Module-logger in elk script:
* import logging
* logger = logging.getLogger(\_\_name\_\_)

**Wanneer loggen?**

* Altijd: start/einde functies, belangrijke beslissingen, fouten.
* Optioneel: debug-informatie, waarschuwingen.

**Logniveaus**

* debug = details
* info = normale status
* warning = opvallend, niet kritisch
* error = fout
* exception = fout mét traceback

**Schrijfstijl**

* Kort en duidelijk:
* logger.info("Bestand geladen: %s", pad)
* logger.error("Kon bestand niet openen: %s", pad)
* Geen overkill.

**Structuurregels**

* Bovenaan script altijd module-logger.
* 2–3 kern-logregels per functie.
* Gebruik markers rond functies.

**Logbeheer**

* logs/ niet committen.
* Oude logs wissen mag; nieuwe worden aangemaakt.
* Analyse → via tools/log\_report.py.

**Testen**

* Run project na wijzigingen.
* Controleer log.txt en log.jsonl.

**Consistentie**

* Altijd logger, nooit print().
* Functionele berichten, geen irrelevante tekst.

**👉 Wat moet ChatGPT doen in een nieuw chatproject?**

1. Altijd eerst beschrijven hoe een nieuw project eruit ziet (structuur, logging, markers).
2. Daarna antwoorden geven volgens deze richtlijnen:
   * Gebruik altijd de vaste formulieren (ADD/REPLACE/DELETE/CREATE).
   * Houd je aan de marker-structuur.
   * Voeg logging toe zoals beschreven.
   * Houd consistentie aan in stijl en markers.
3. Nooit vergeten: in een nieuw chatproject start ChatGPT met leeg geheugen → deze samenvatting is de basis om te begrijpen hoe je project verder moet worden opgebouwd.