

Red means uncertain/major revision needed
Pink means it is a suggestion
but maybe not necessary

Type is a reserved keyword in Python

«Enumeration»
WeekDay
Monday: 0
Tuesday: 1
Wednesday: 2
Thursday: 3
Friday: 4
Saturday: 5
Sunday: 6

«Enumeration»
QuestionStatus
unanswered: 0
answered: 1
snoozed: 2
cancelled: 3

Variable
- var_type: Class
- name: str
+ get_type(): Class
+ get_name(): str

DatabaseWriter
- conn: Connection
- variables: Set[Variable]
+ get_variables() -> Iterable[Variable]
+ execute_sql(sql_command: str)
+ execute_query(sql_command: str): List[Tuple]

«Abstract»
Command
- is_executed: bool = False
- db_writer: DatabaseWriter
- execute_sql_code: str
- laset_result: List[Tuple] = None
+ execute()
+ get_is_executed(): bool

«Abstract»
ReversibleCommand
- is_executed: bool = False
- db_writer: DatabaseWriter
- id: int
+ get_is_executed(): bool
+ get_id(): int
+ execute()
+ undo()

InsertCommand
- is_executed: bool = False
- db_writer: DatabaseWriter
- id: in
- variable: Variable
- timestamp: int
- new_value: Any
+ get_is_executed(): bool
+ get_id()
+ execute()
+ undo()

Answer
- id: str
- timestamp: float
- variable: Variable
- value: variable.var_type
+ get_id(): int
+ get_value(): Any
+ extract_command(db_writer: DatabaseWriter):
 InsertCommand

BackendQuestioner
- executed_commands: List[Command]
- undone_commands: List[Command]
- schedule: QuestionSchedule
- db_writer: DatabaseWriter
+ get_active_questions(): Set[Question]
+ mainloop(pipe: multiprocessing.Pipe)
 + add_new_variable(var: Variable)
 + store_answer(answer: Answer)
 + register_incoming_answers(): Iterable[Answer]
 + send_questions(questions: Set[Question])

QuestionSchedule
- active_occurrences: Dict[int, QuestionOccurrence]
- snoozed_occurrences: Dict[int, QuestionOccurrence]
- finished_occurrences: Dict[int, QuestionOccurrence]
- prev_refresh_timestamp: float
- timeout_time: float
- question_sets: Set[ReoccurringQuestionSet]
- formulator: Formulator
+ get_active_occurrences(): Set[QuestionOccurrence]
+ register_answered(id: int)

QuestionOccurrence
- status: QuestionStatus
- questions: Question
- snooze_time: int = 0
- id: int
+ get_id(): int
+ get_status(): QuestionStatus
+ set_status(new_status: QuestionStatus)
+ snooze(minutes: int)

Question
- content: str
- variable: Variable
+ get_content(): str
+ get_variable(): Variable

RecurringQuestionSet
- days: Set[WeekDay]
- hour: int
- minutes: int
- variables: Set[Variable]
+ get_days(): Set[WeekDay]
+ get_hour(): int
+ get_minutes(): int
+ get_variables(): Set[Variable]
+ is_due(current_time: float, previous_check: float): bool
+ generate_questions(formulator: Formulator): Set[QuestionOccurrence]

Formulator
+ formulate_question(variable: Variable): str

ComparisonRule
- var_2: Variable
- relation: Union[==, >, >=]

RelativeChangeRule
- period: float
- relative_magnitude: float

ThresholdRule
- threshold: float

ANDCompositeRule

ORCompositeRule

«Abstract»
Rule
- var: Variable
- query: QueryCommand
- fireable: bool = False
+ get_var(): Iterable[Variable]
+ check_fireable(): bool
+ set_fired()
+ report_rows_of_interest(): List[Tuple]

QueryCommand
- is_executed: bool = False
- database: DatabaseWriter
- execute_sql_code: str
- laset_result: List[Tuple] = None
+ execute(): List[Tuple]
+ get_last_result(): List[Tuple]
+ get_is_executed(): bool

OutputInvoker
- rules: Iterable[Rule]
+ mainloop(pipe: multiprocessing.Pipe)
 + find_invoked_rules(): Iterable[Rule]
 + invoke_rules():

OutputTextGenerator
- state: Tensor

«Abstract»
torch.nn.Module
+ get_named_parameters(): Iterable[Tuple]
+ get_parameters(): Iterable[Parameter]
+ forward(x: Tensor): Tensor

After firing,
only becomes fireable again
when new changes occur in the
database that make the rule fire.
Otherwise the same output would
be generated indifferently.

Fire when a Variable made a relat
large change (> magnitude)
w.r.t. the average value during
the past [period] time.

Only allow sets of questions to be done together?
(Can always just use a single question in QuestionOccurrence,
but we could enforce this)

Status superfluous?
Snooze needs revision?

Questions are stores in a dict
that maps the ID of an Occurrence
to the QuestionOccurrence itself.

We will probably need to use multiprocessing.
Perhaps a 'Pipe' to send data between processes.
Here it will be used to exchange Questions and Answers.
The BackendQuestioner will be able to match a Question
to its Answer via their ID.

For commands such as undo or adding a variable
we can make new classes. Then the BackendQuestioner
can just extract stuff from the pipe,
and check the type of the object (answer, new var, undo)
to figure out what needs to be done with it.

Output Production Process

Data Gather Process

GUI Process

GUI