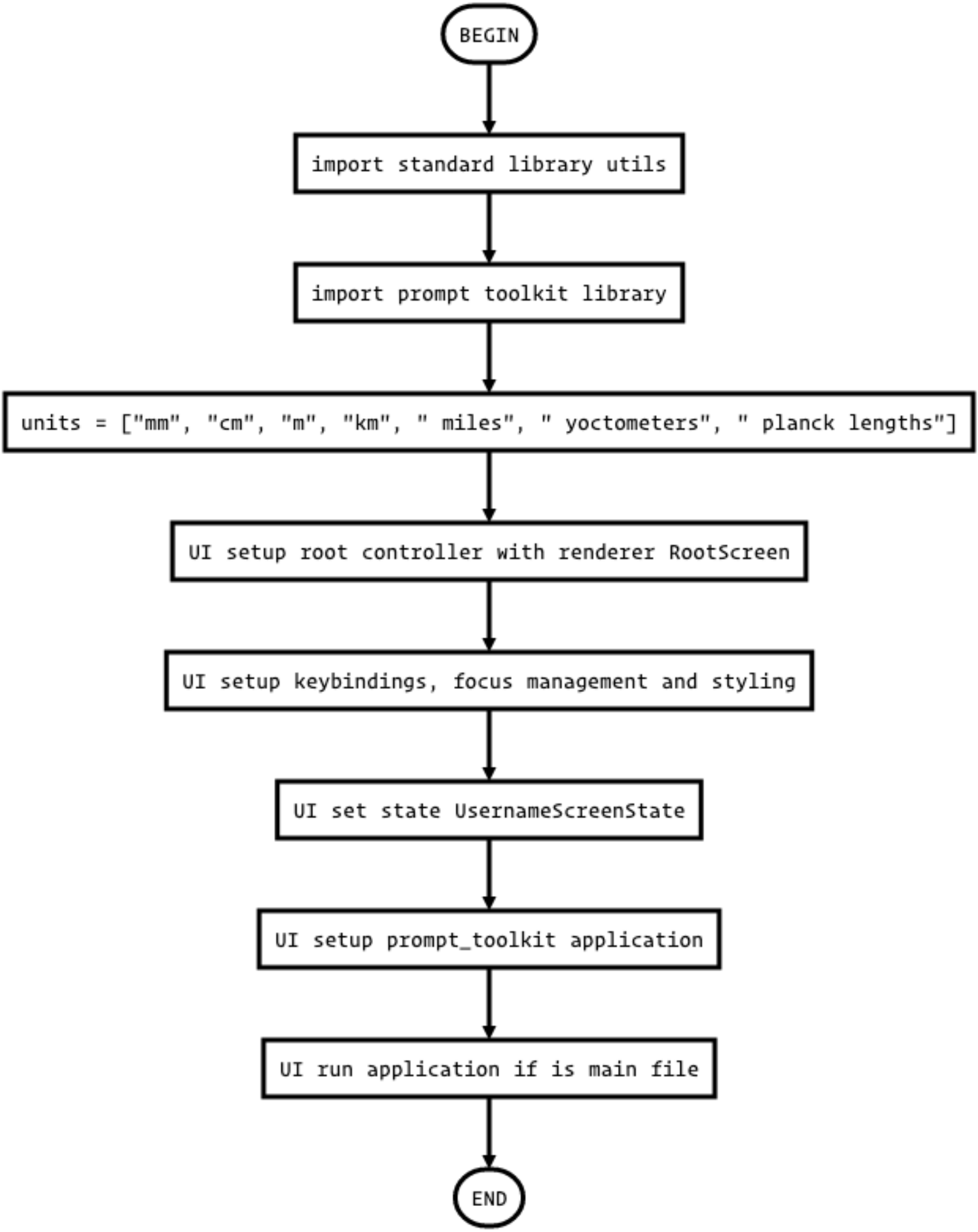
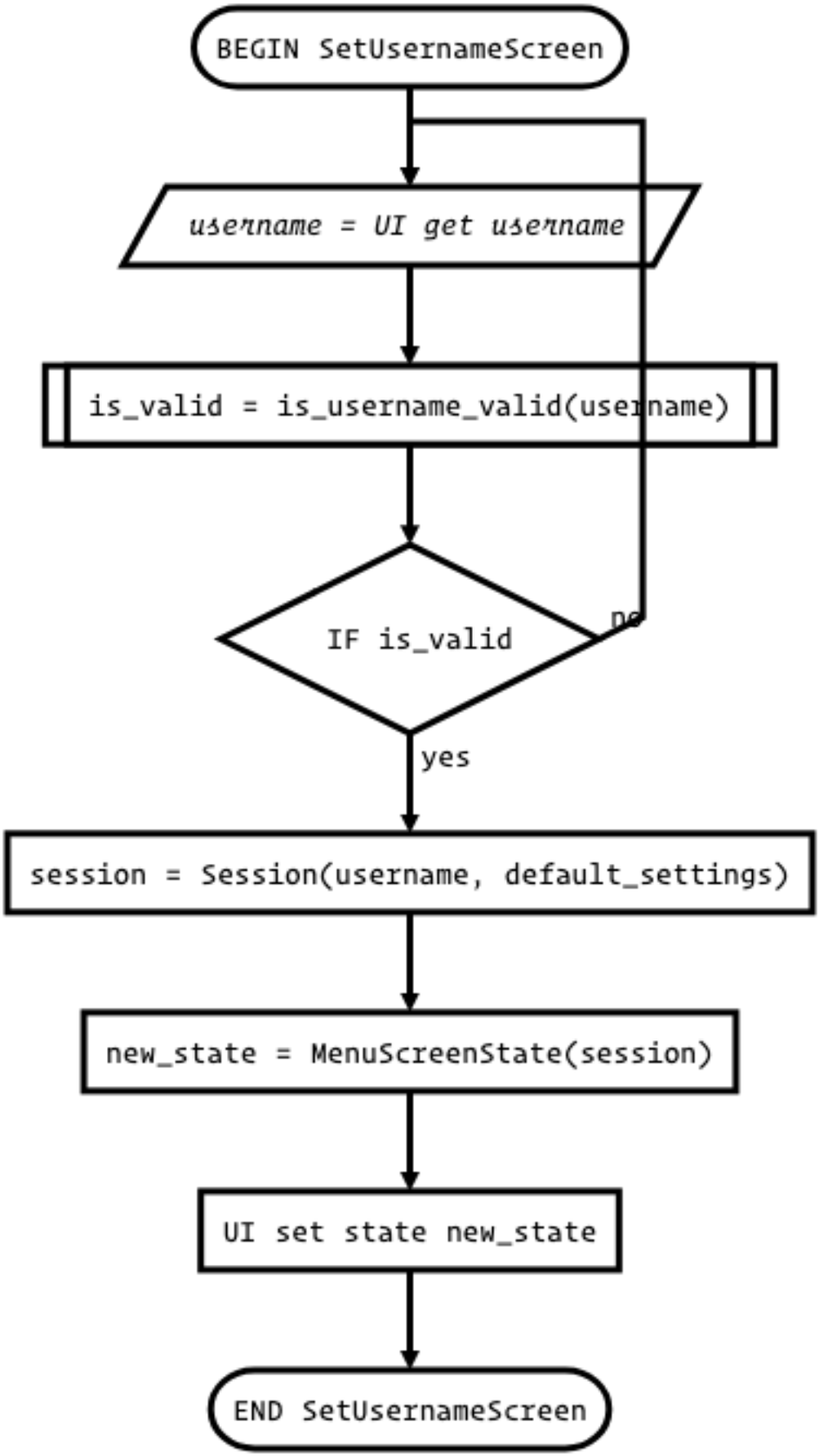
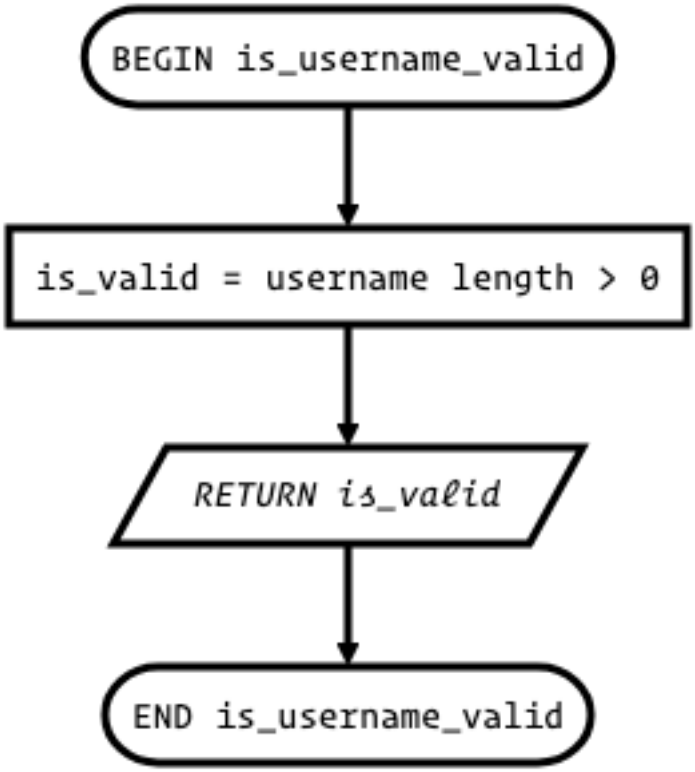
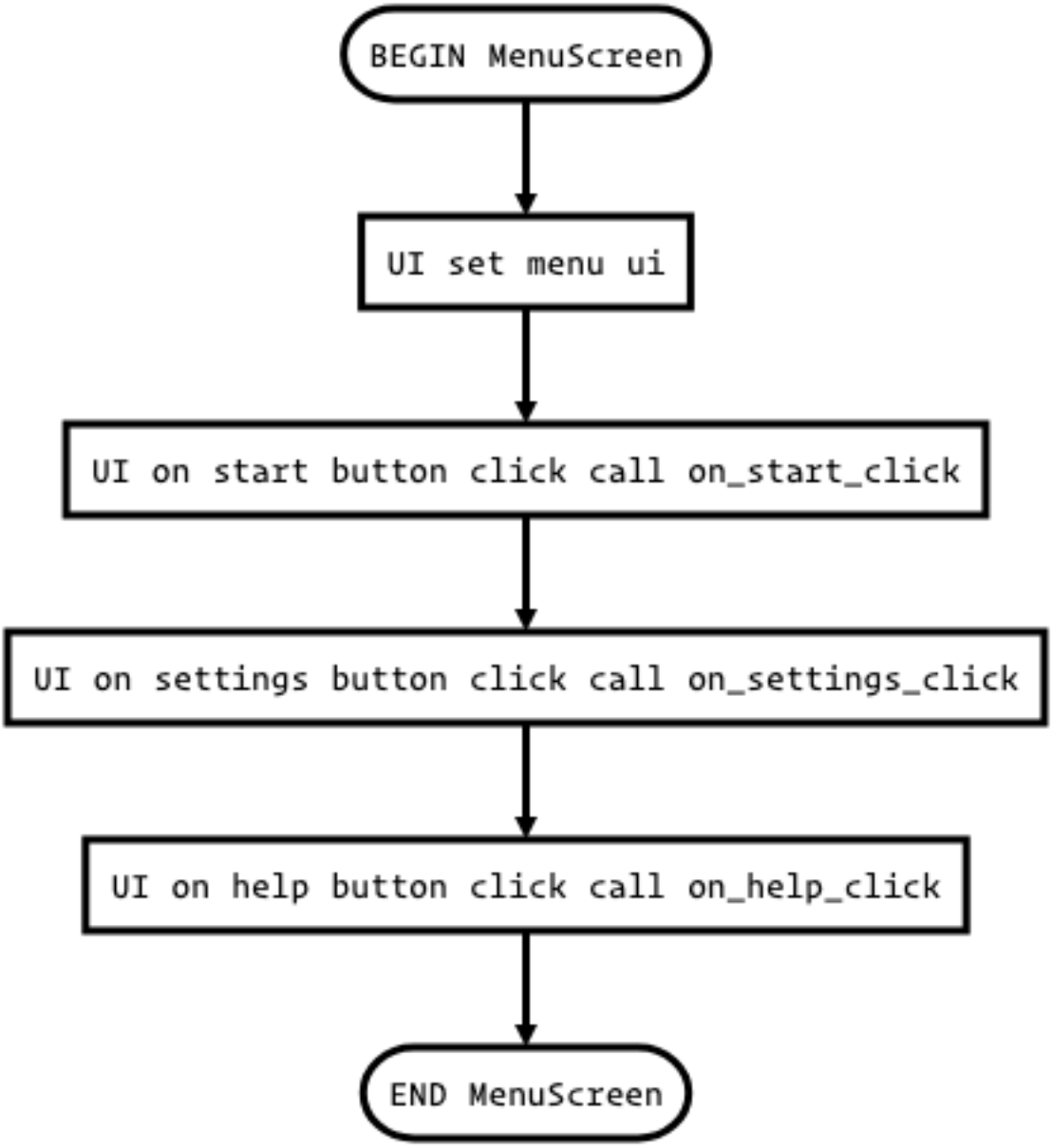
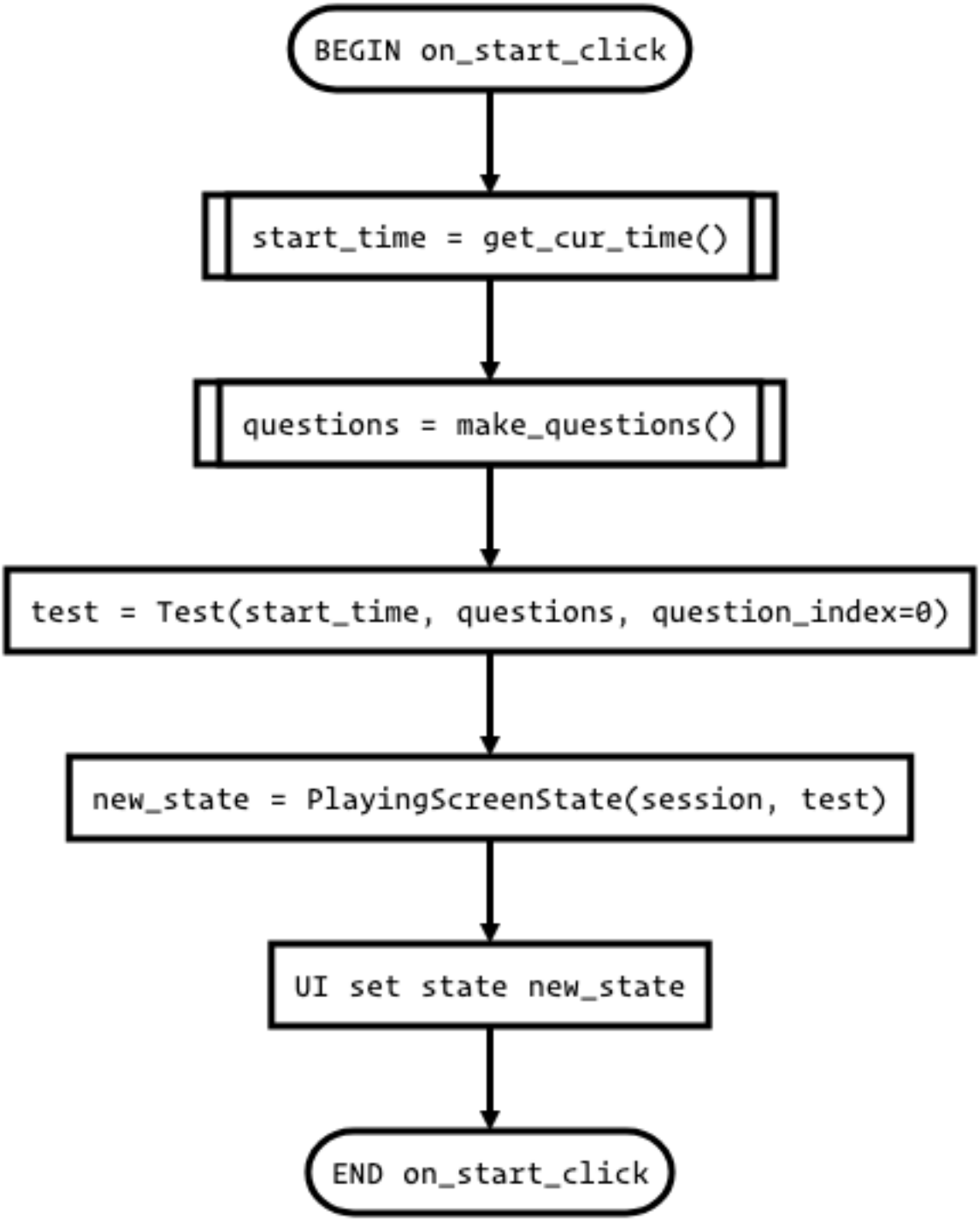
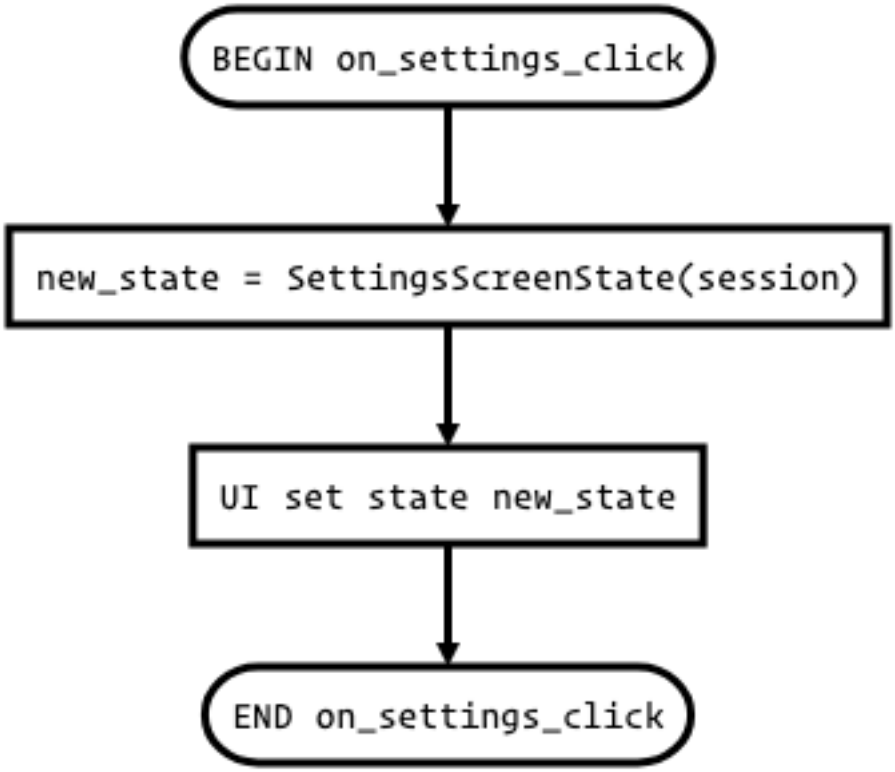

BEGIN
import standard library utils
import prompt toolkit library
units = ["mm", "cm", "m", "km", " miles", " yoctometers", " planck lengths"]
UI setup root controller with renderer RootScreen
UI setup keybindings, focus management and styling
UI set state UsernameScreenState
UI setup prompt_toolkit application
UI run application if is main file
END

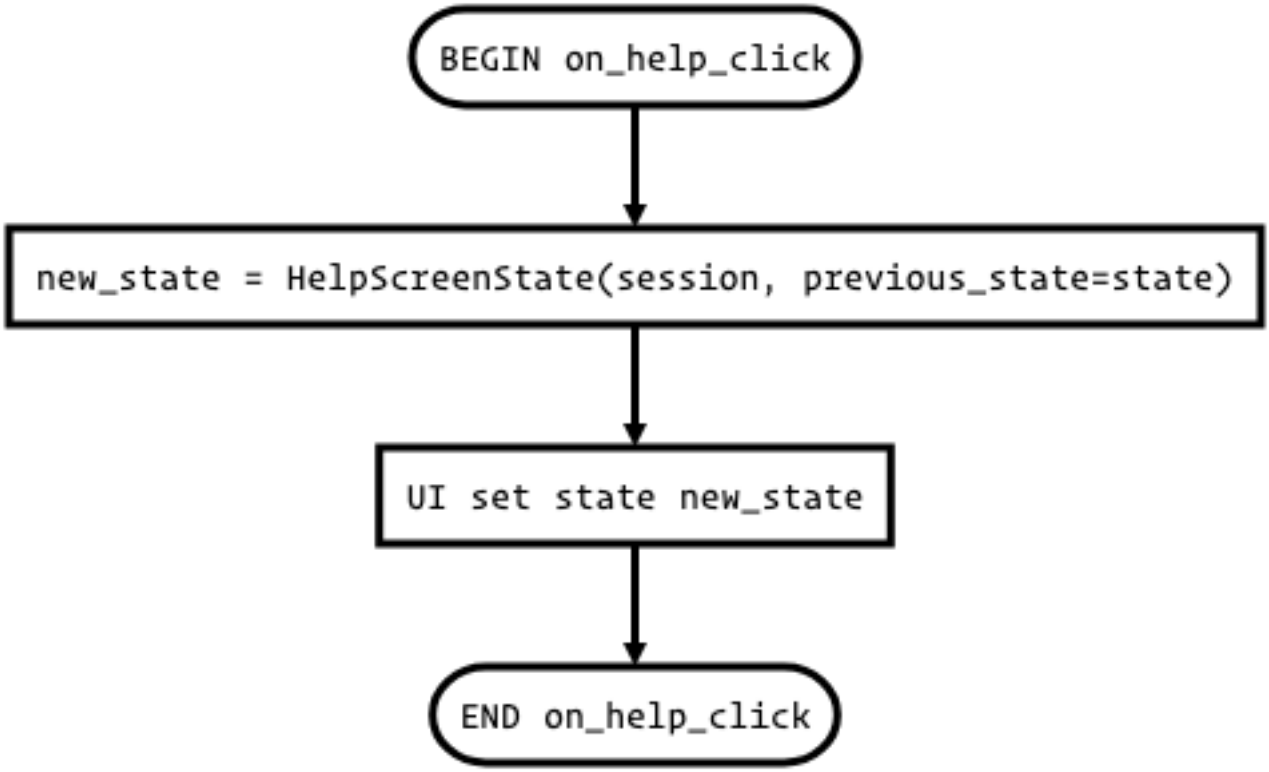

BEGIN SetUsernameScreen
username = UI get username
is_valid = is_username_valid(username)
IF is_valid
no
yes
session = Session(username, default_settings)
new_state = MenuScreenState(session)
UI set state new_state
END SetUsernameScreen

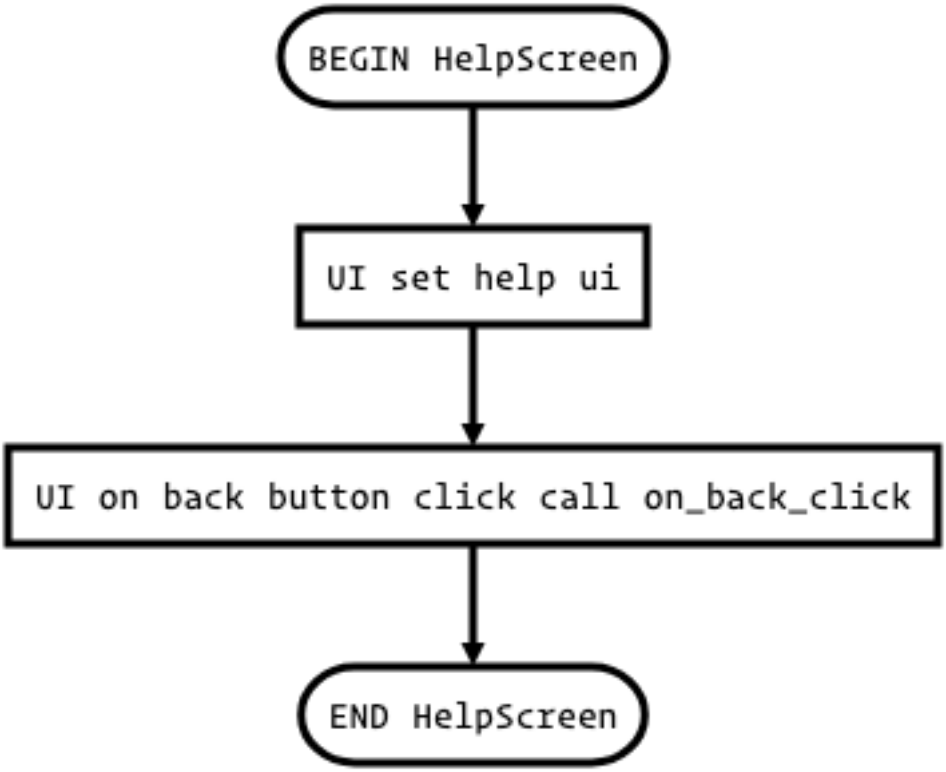

BEGIN is_username_valid
is_valid = username length > 0
RETURN is_valid
END is_username_valid

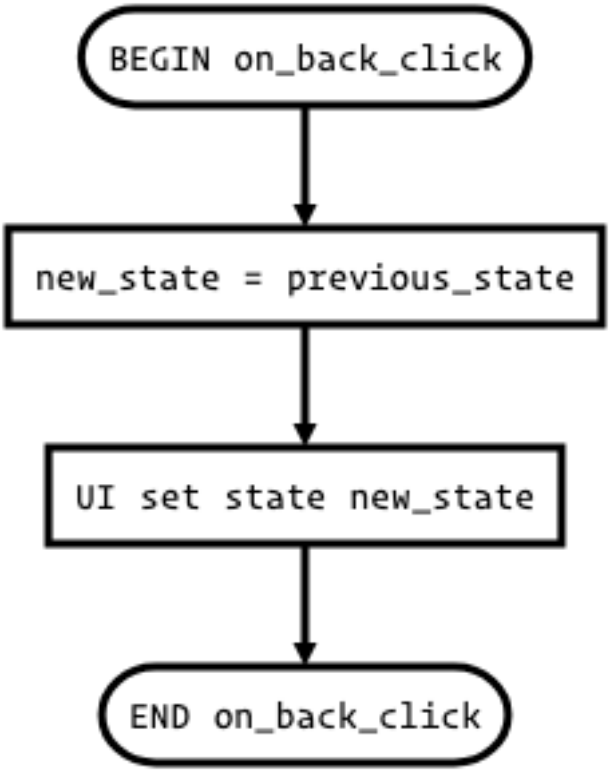

BEGIN MenuScreen
UI set menu ui
UI on start button click call on_start_click
UI on settings button click call on_settings_click
UI on help button click call on_help_click
END MenuScreen

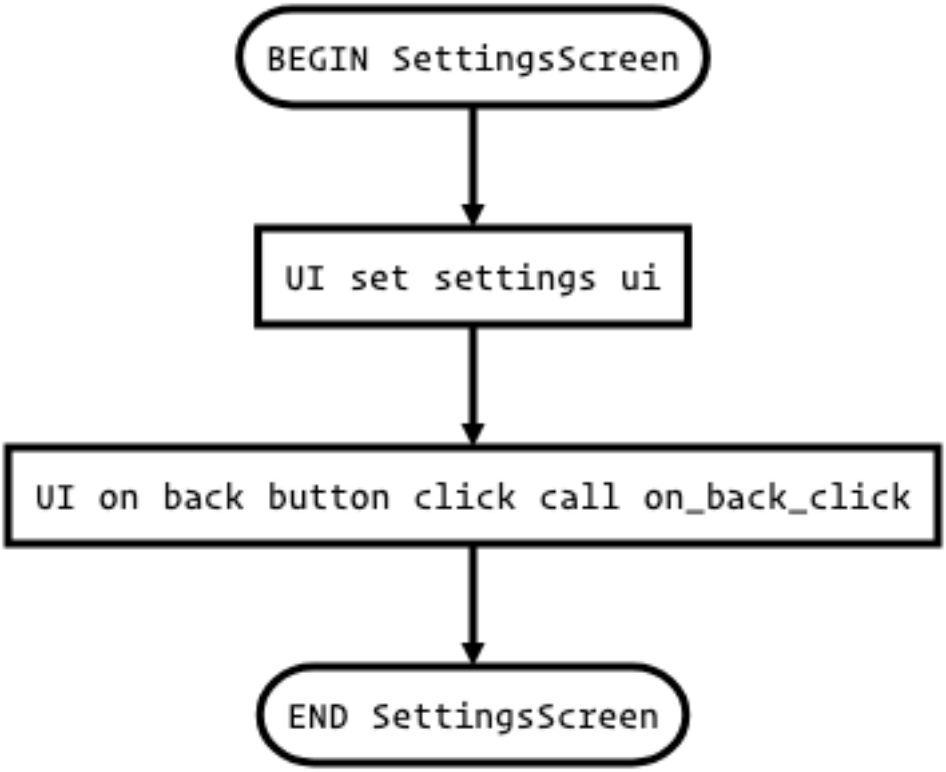

BEGIN on_start_click
start_time = get_cur_time()
questions = make_questions()
test = Test(start_time, questions, question_index=0)
new_state = PlayingScreenState(session, test)
UI set state new_state
END on_start_click

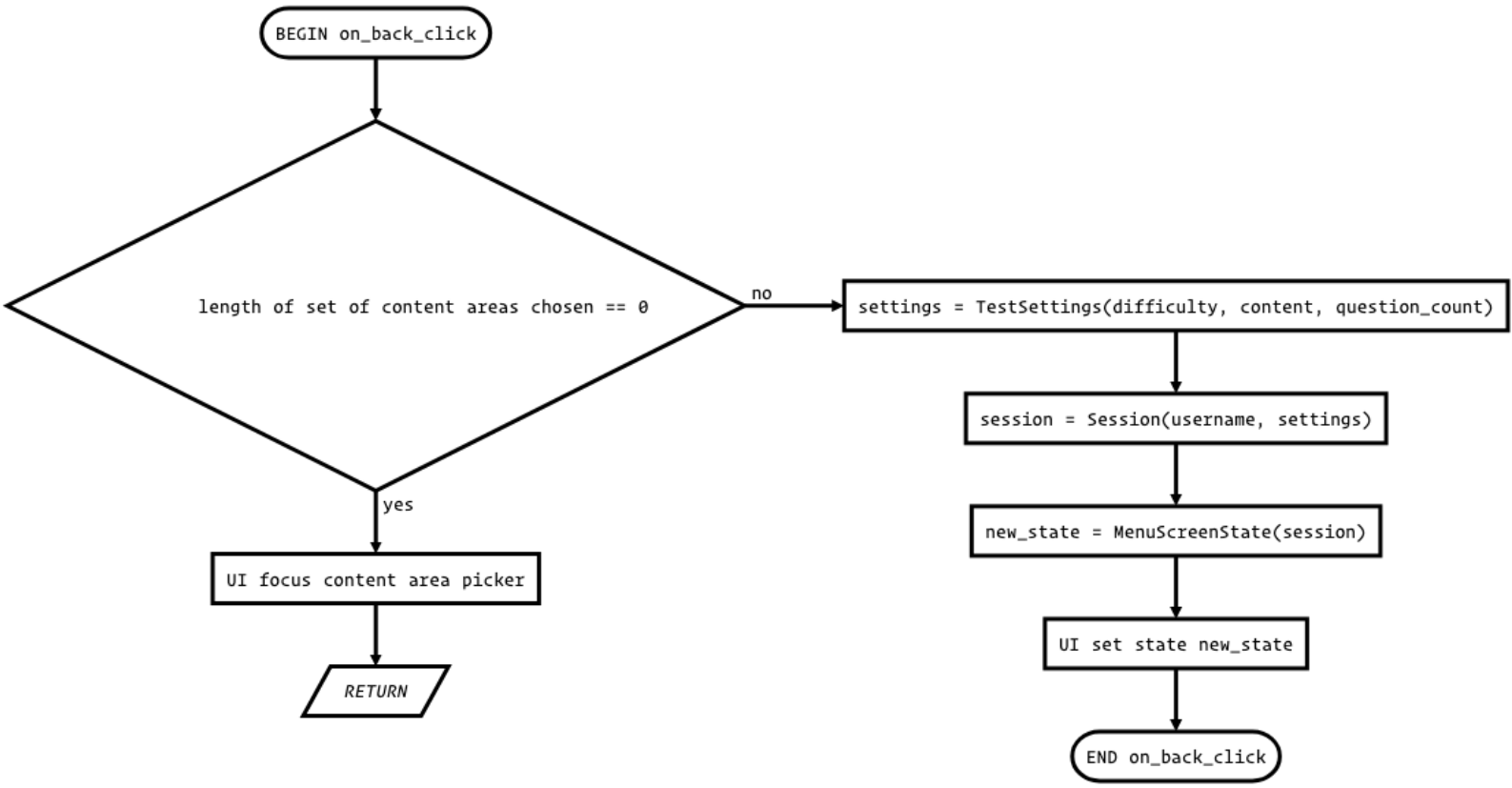

BEGIN on_settings_click
new_state = SettingsScreenState(session)
UI set state new_state
END on_settings_click

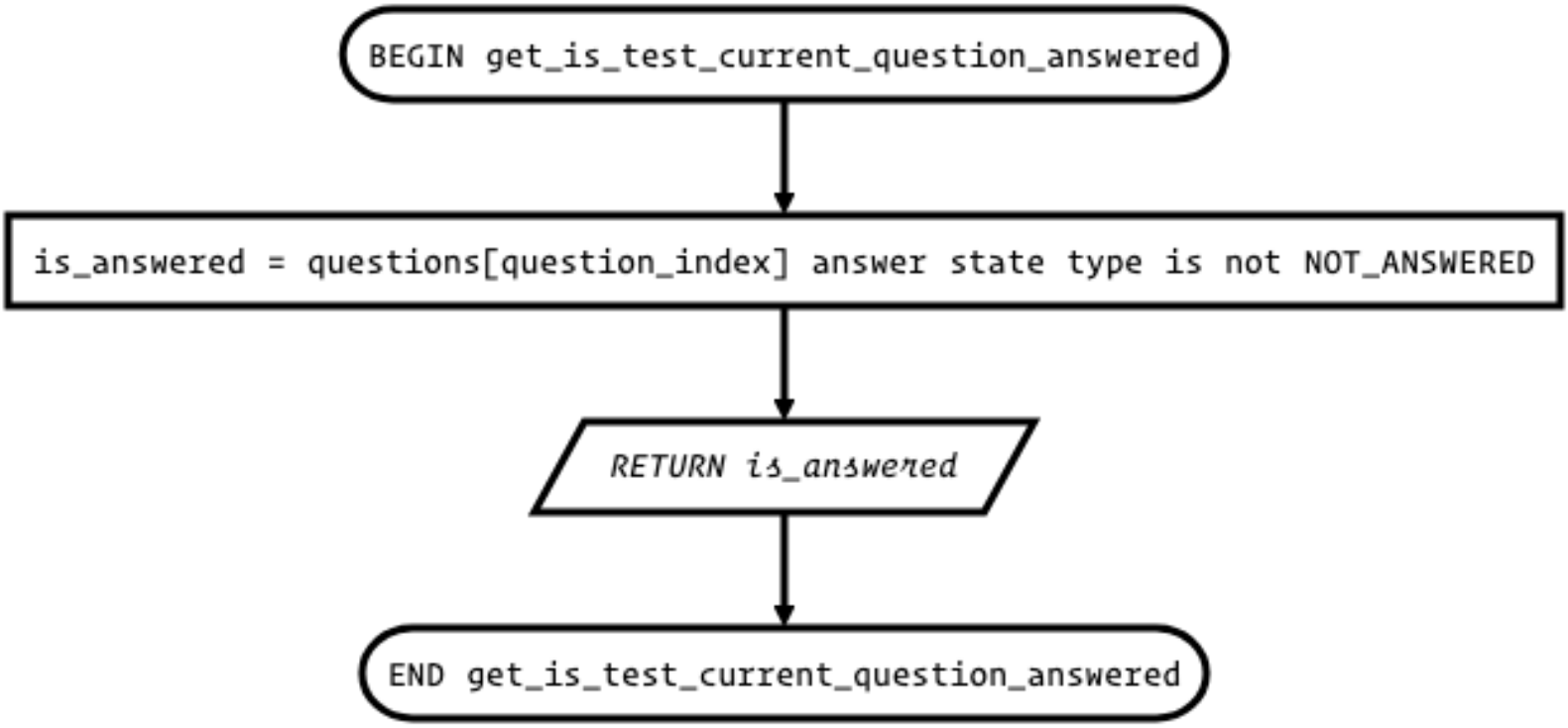

BEGIN on_help_click
new_state = HelpScreenState(session, previous_state=state)
UI set state new_state
END on_help_click

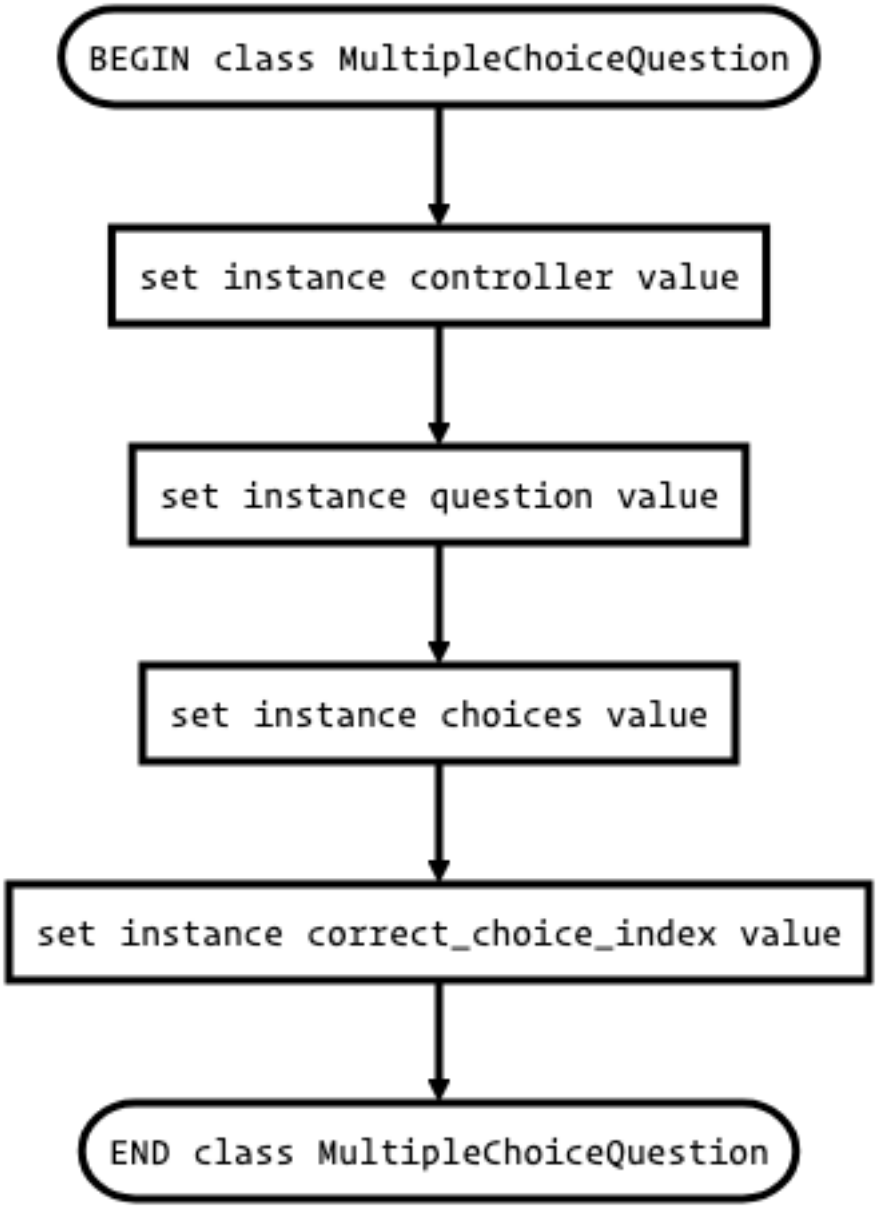

BEGIN HelpScreen
UI set help ui
UI on back button click call on_back_click
END HelpScreen

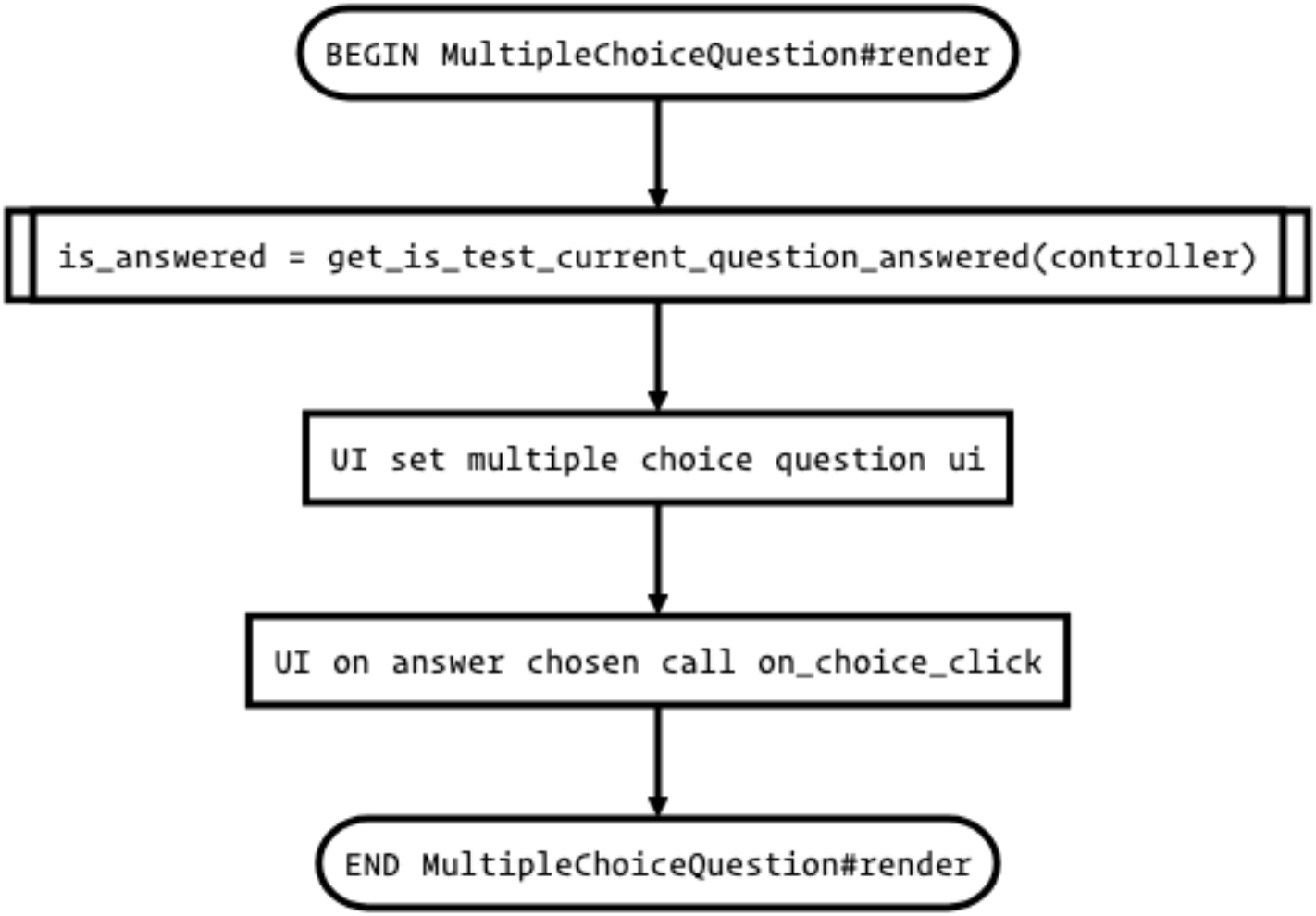

BEGIN on_back_click
new_state = previous_state
UI set state new_state
END on_back_click

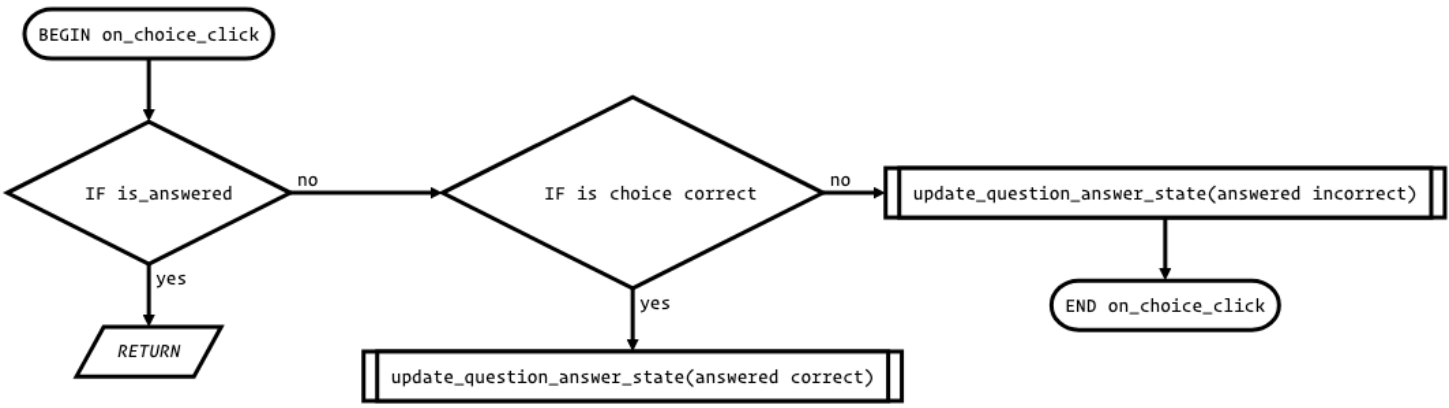

BEGIN SettingsScreen
UI set settings ui
UI on back button click call on_back_click
END SettingsScreen

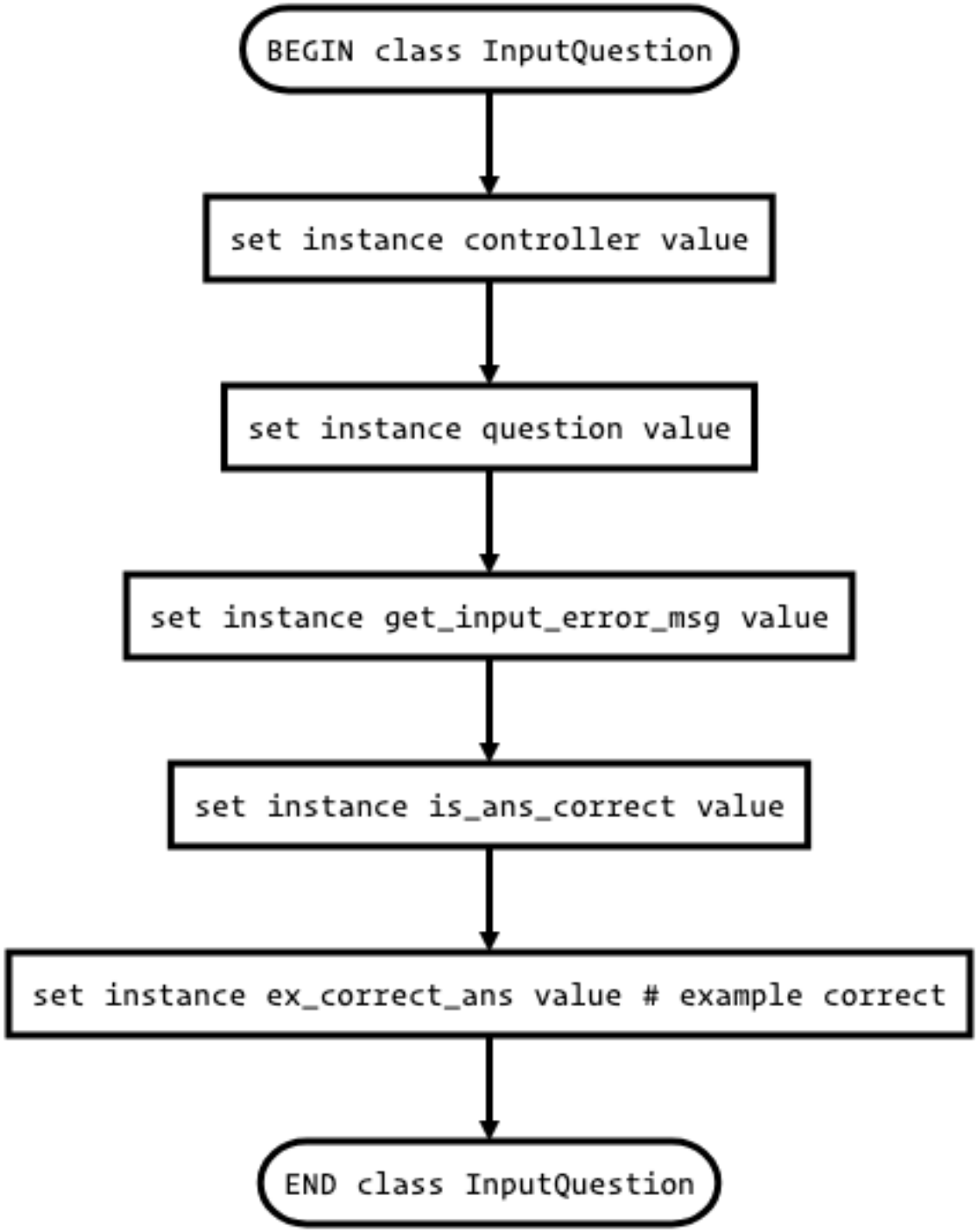

BEGIN on_back_click
length of set of content areas chosen == 0
no
yes
UI focus content area picker
RETURN
settings = TestSettings(difficulty, content, question_count)
session = Session(username, settings)
new_state = MenuScreenState(session)
UI set state new_state
END on_back_click

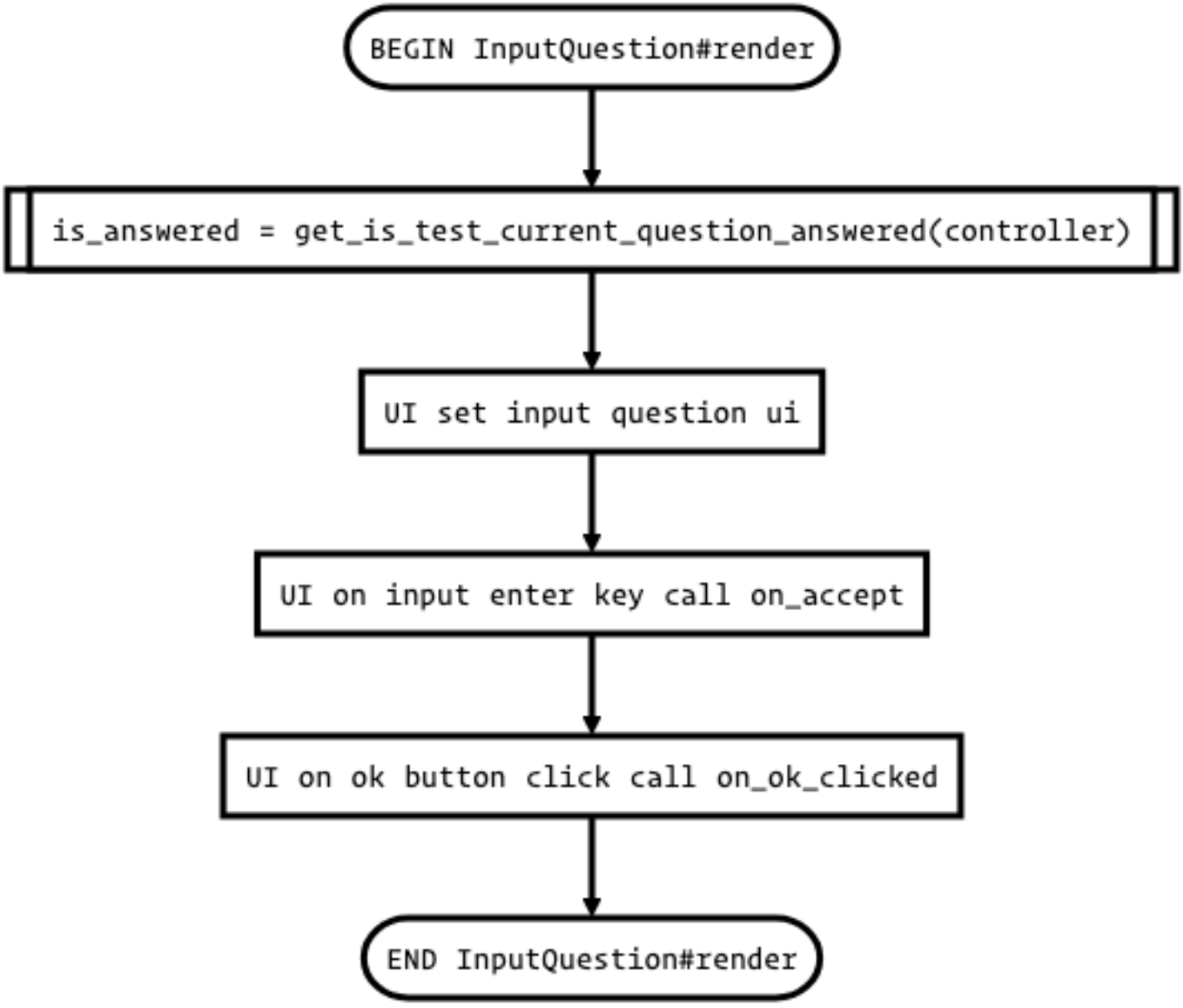

BEGIN get_is_test_current_question_answered
is_answered = questions[question_index] answer state type is not NOT_ANSWERED
RETURN is_answered
END get_is_test_current_question_answered

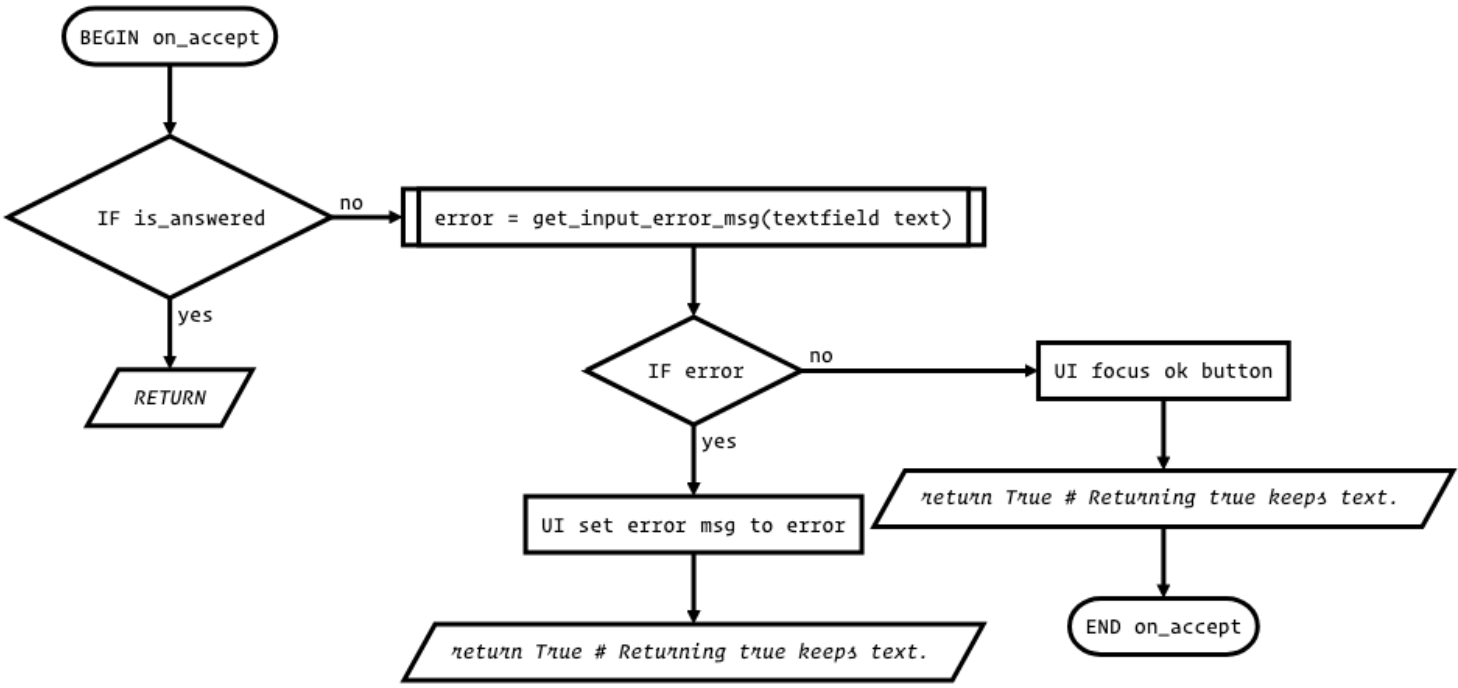

BEGIN class MultipleChoiceQuestion
set instance controller value
set instance question value
set instance choices value
set instance correct_choice_index value
END class MultipleChoiceQuestion

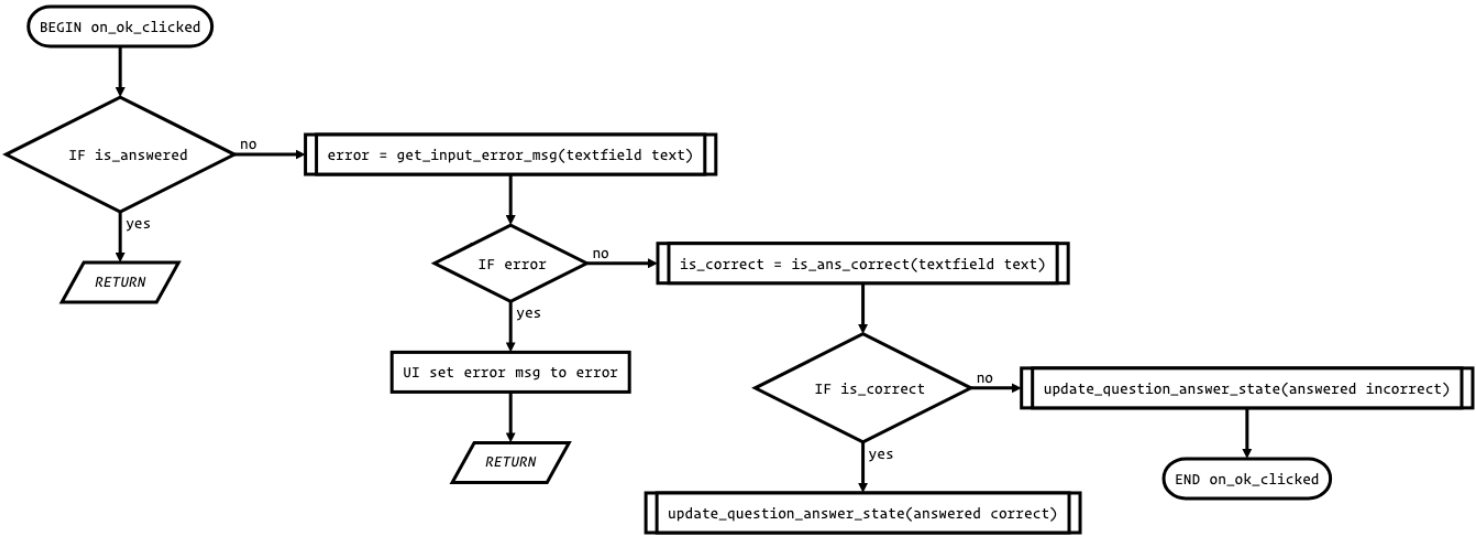

BEGIN MultipleChoiceQuestion#render
is_answered = get_is_test_current_question_answered(controller)
UI set multiple choice question ui
UI on answer chosen call on_choice_click
END MultipleChoiceQuestion#render

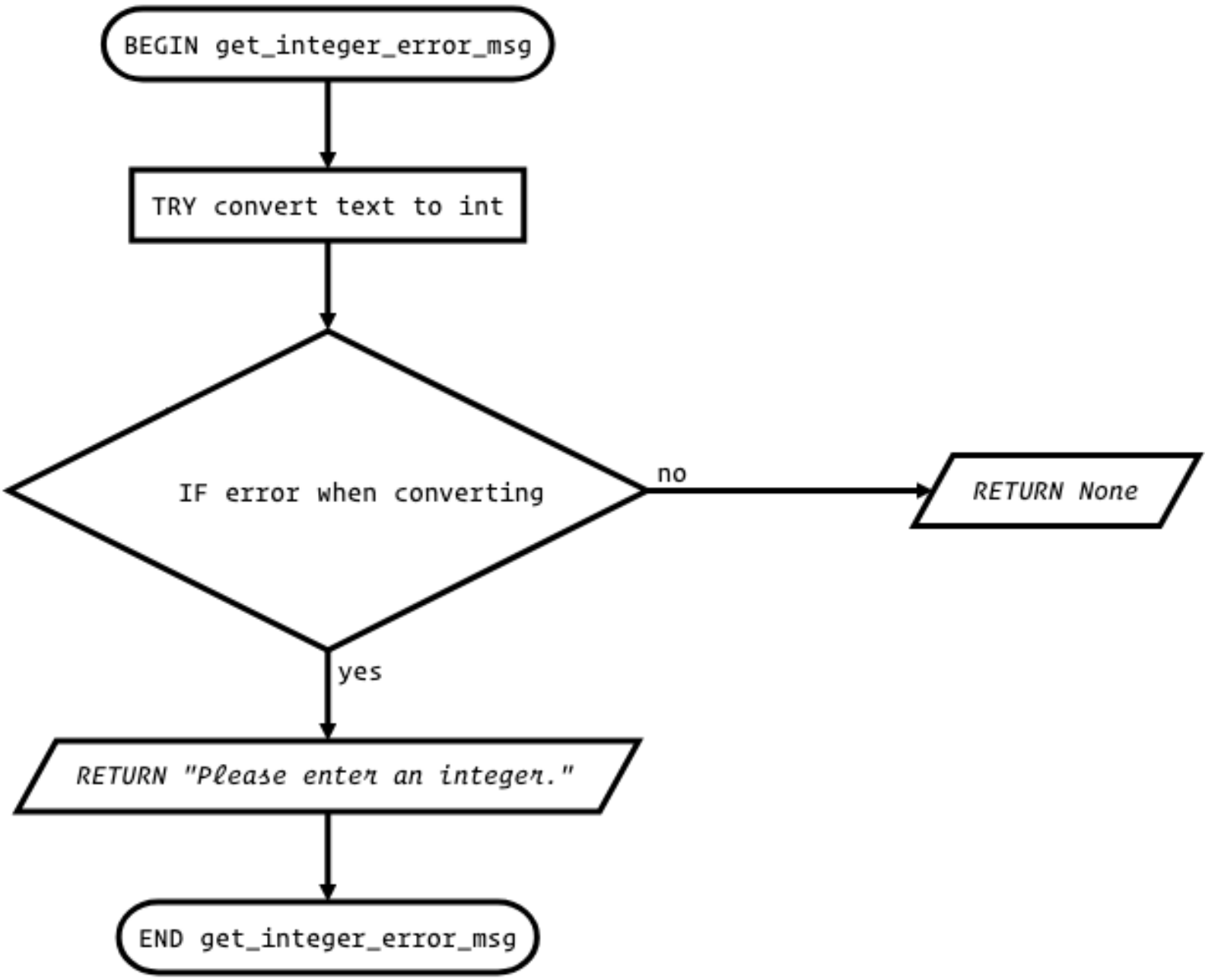

BEGIN on_choice_click
IF is_answered
no
yes
RETURN
IF is choice correct
no
yes
update_question_answer_state(answered incorrect)
END on_choice_click
update_question_answer_state(answered correct)

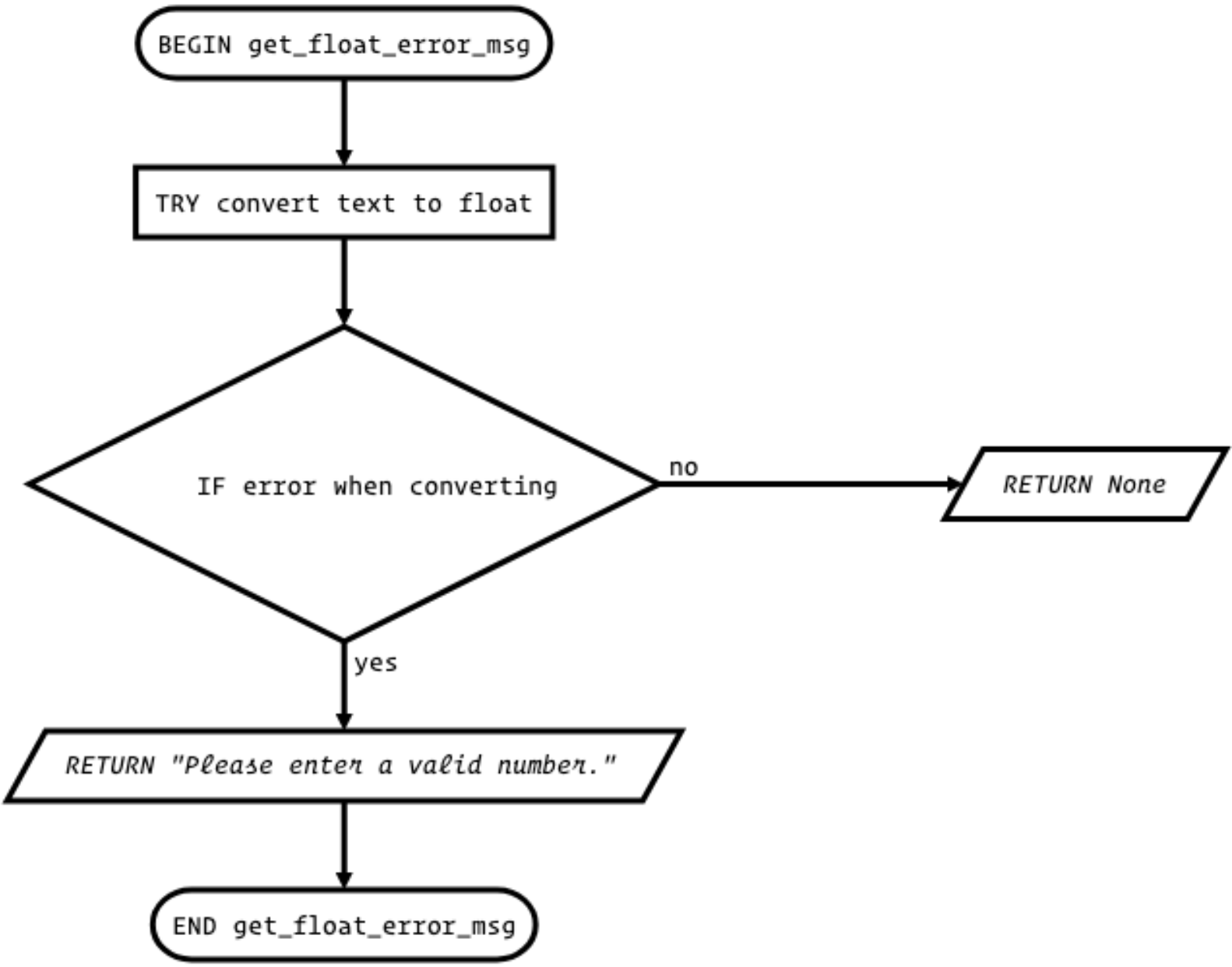

BEGIN class InputQuestion
set instance controller value
set instance question value
set instance get_input_error_msg value
set instance is_ans_correct value
set instance ex_correct_ans value # example correct
END class InputQuestion

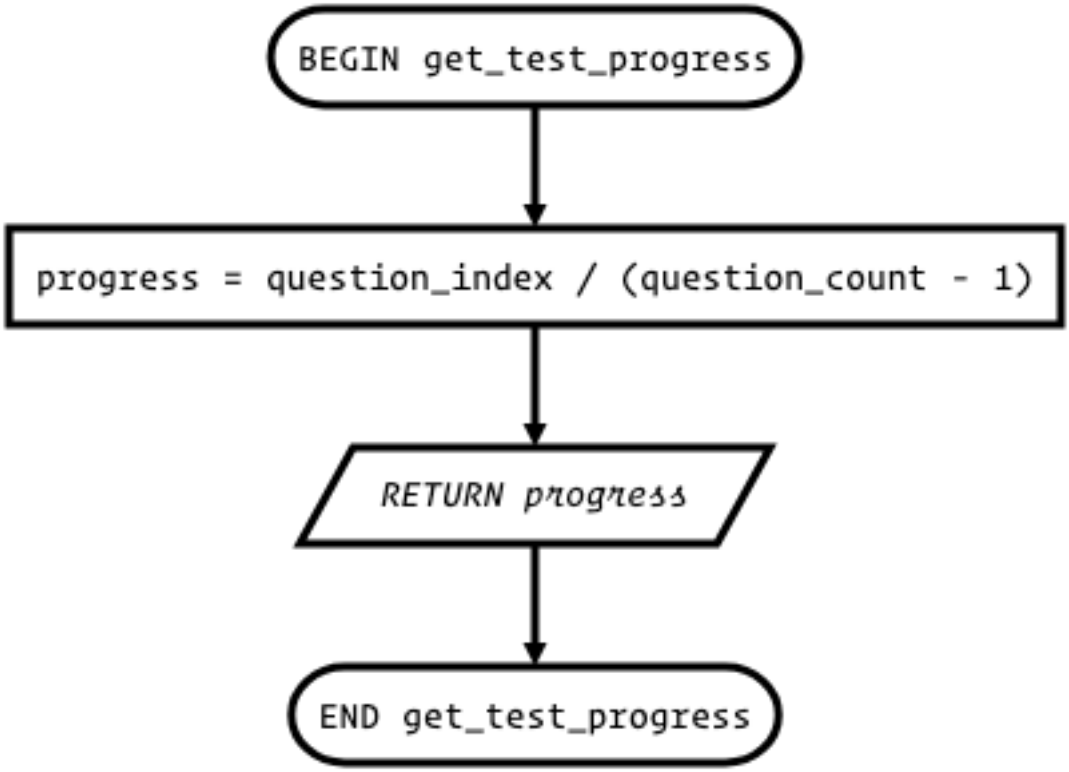

BEGIN InputQuestion#render
is_answered = get_is_test_current_question_answered(controller)
UI set input question ui
UI on input enter key call on_accept
UI on ok button click call on_ok_clicked
END InputQuestion#render

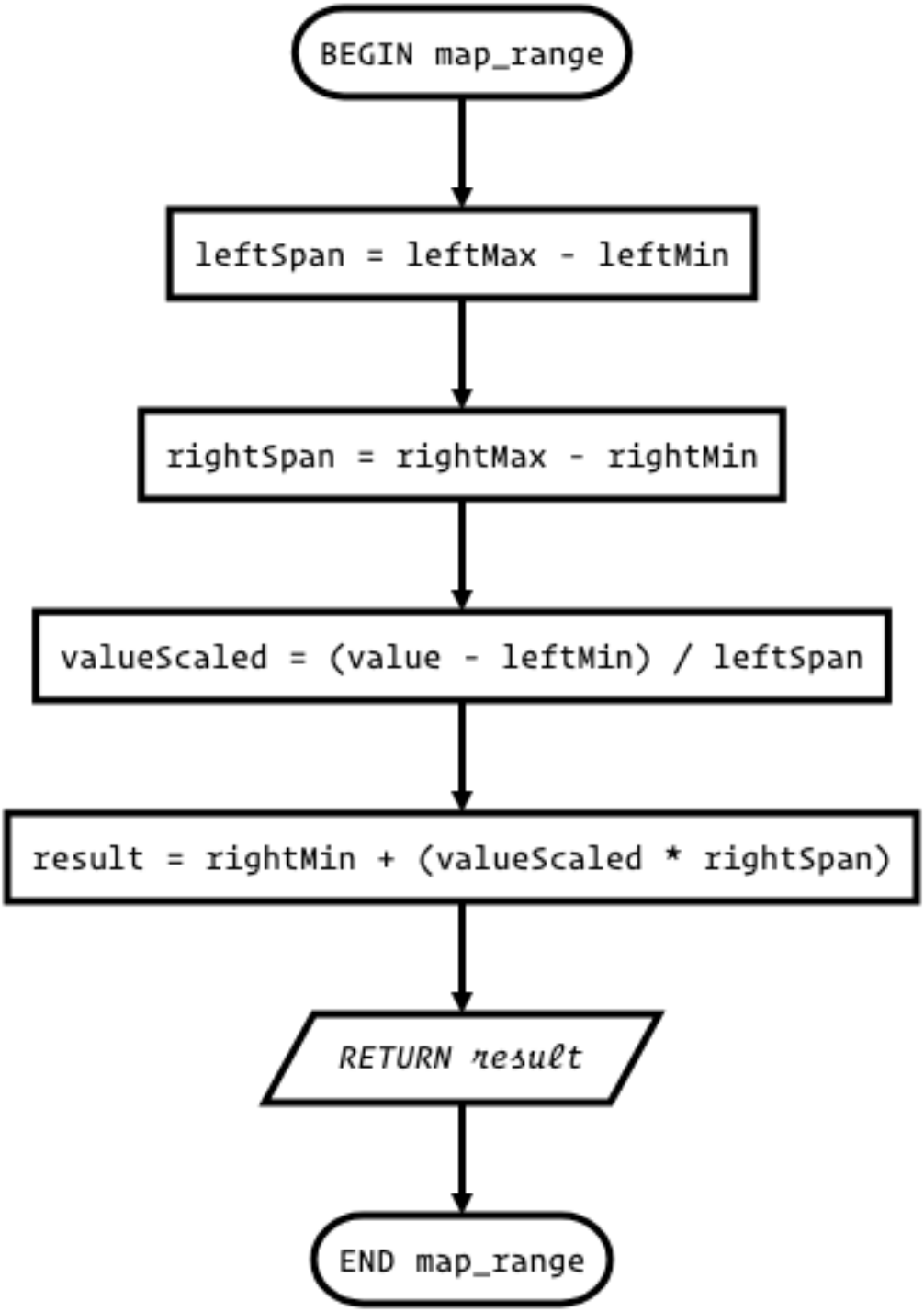

BEGIN on_accept
IF is_answered
no
error = get_input_error_msg(textfield text)
yes
RETURN
IF error
no
UI focus ok button
yes
return True # Returning true keeps text.
UI set error msg to error
END on_accept
return True # Returning true keeps text.

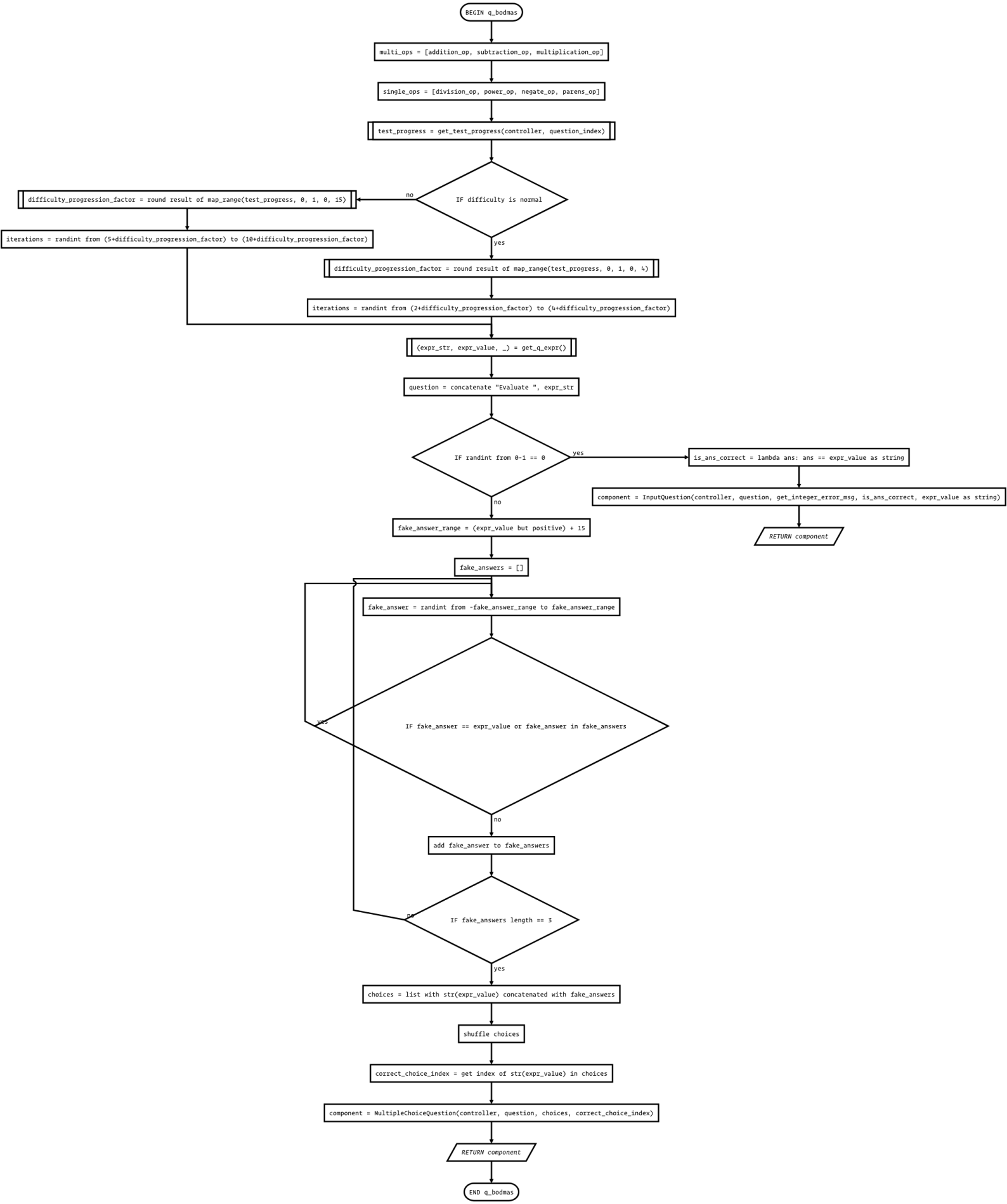

BEGIN on_ok_clicked
IF is_answered
no
yes
RETURN
error = get_input_error_msg(textfield text)
IF error
no
yes
UI set error msg to error
RETURN
is_correct = is_ans_correct(textfield text)
IF is_correct
no
yes
update_question_answer_state(answered incorrect)
END on_ok_clicked
update_question_answer_state(answered correct)

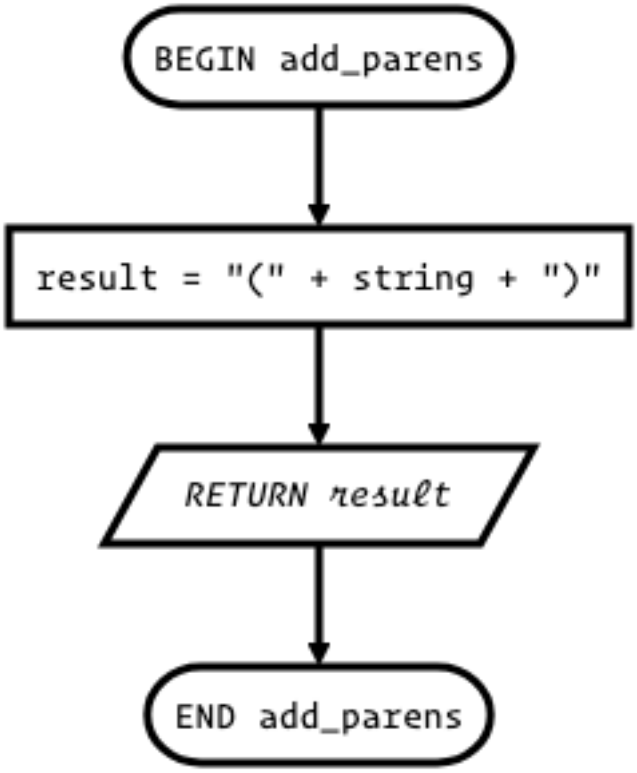

BEGIN get_integer_error_msg
TRY convert text to int
IF error when converting
no
RETURN None
yes
RETURN "Please enter an integer."
END get_integer_error_msg

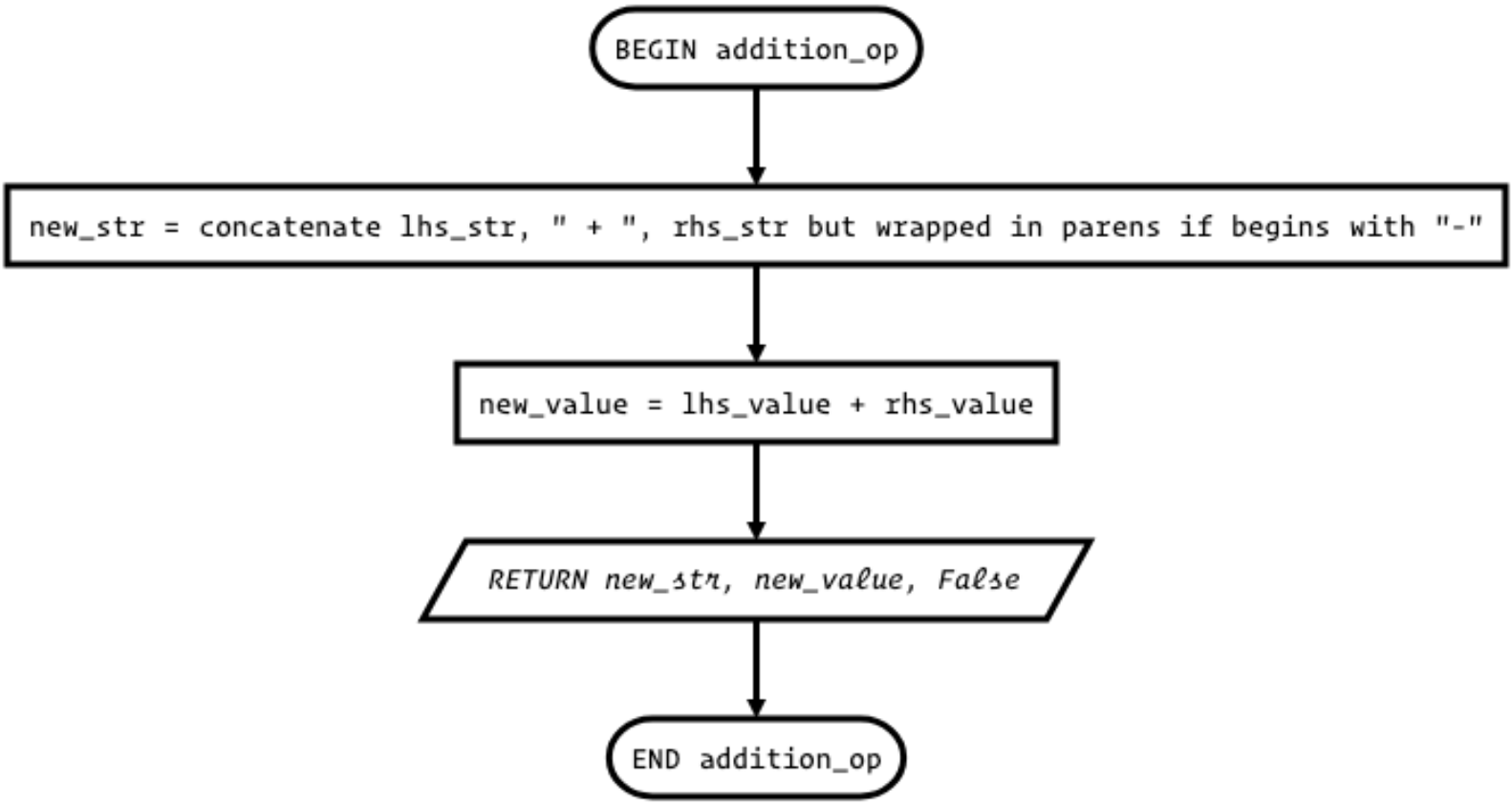

BEGIN get_float_error_msg
TRY convert text to float
IF error when converting
no
RETURN None
yes
RETURN "Please enter a valid number."
END get_float_error_msg

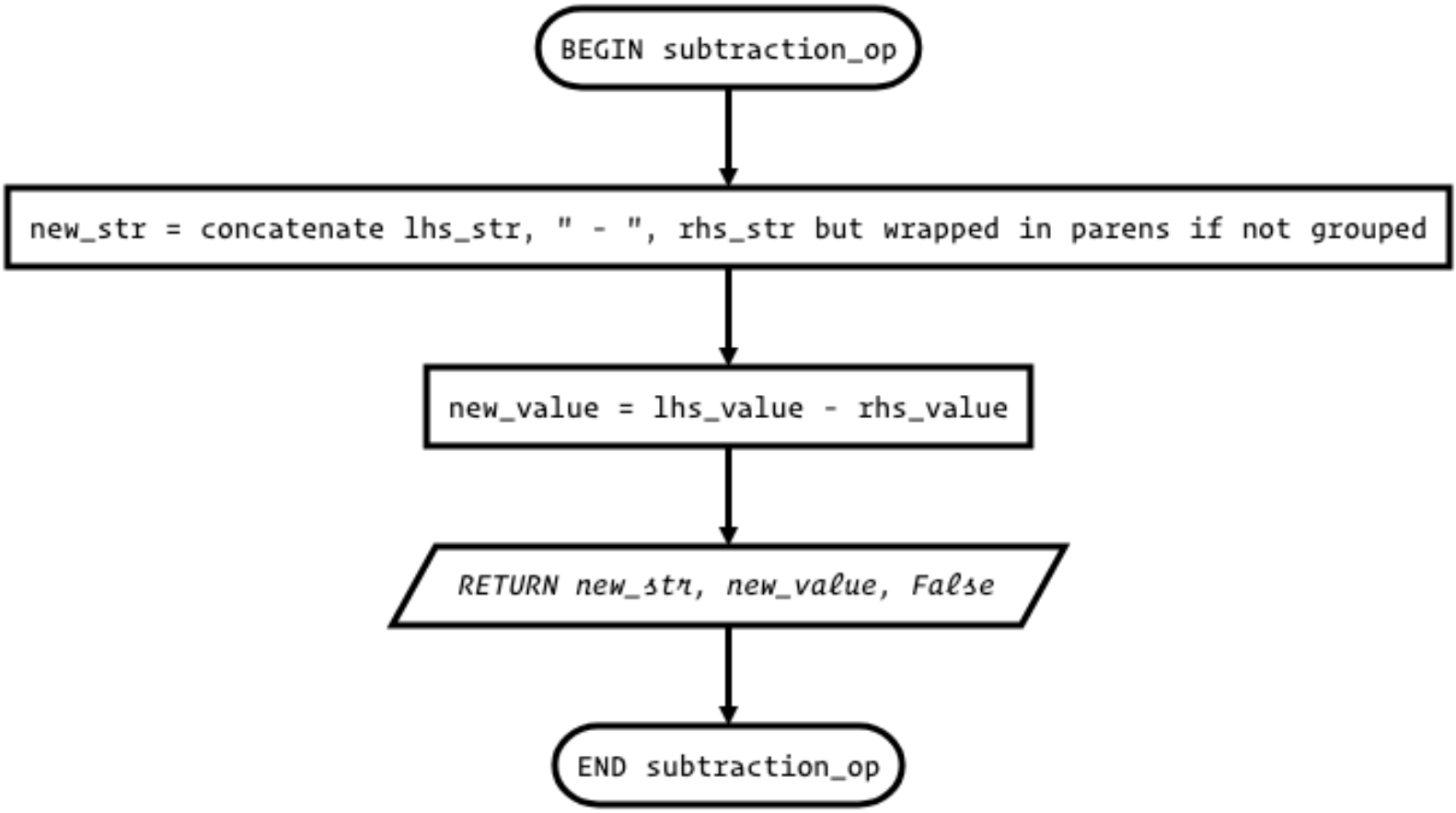

BEGIN get_test_progress
progress = question_index / (question_count - 1)
RETURN progress
END get_test_progress

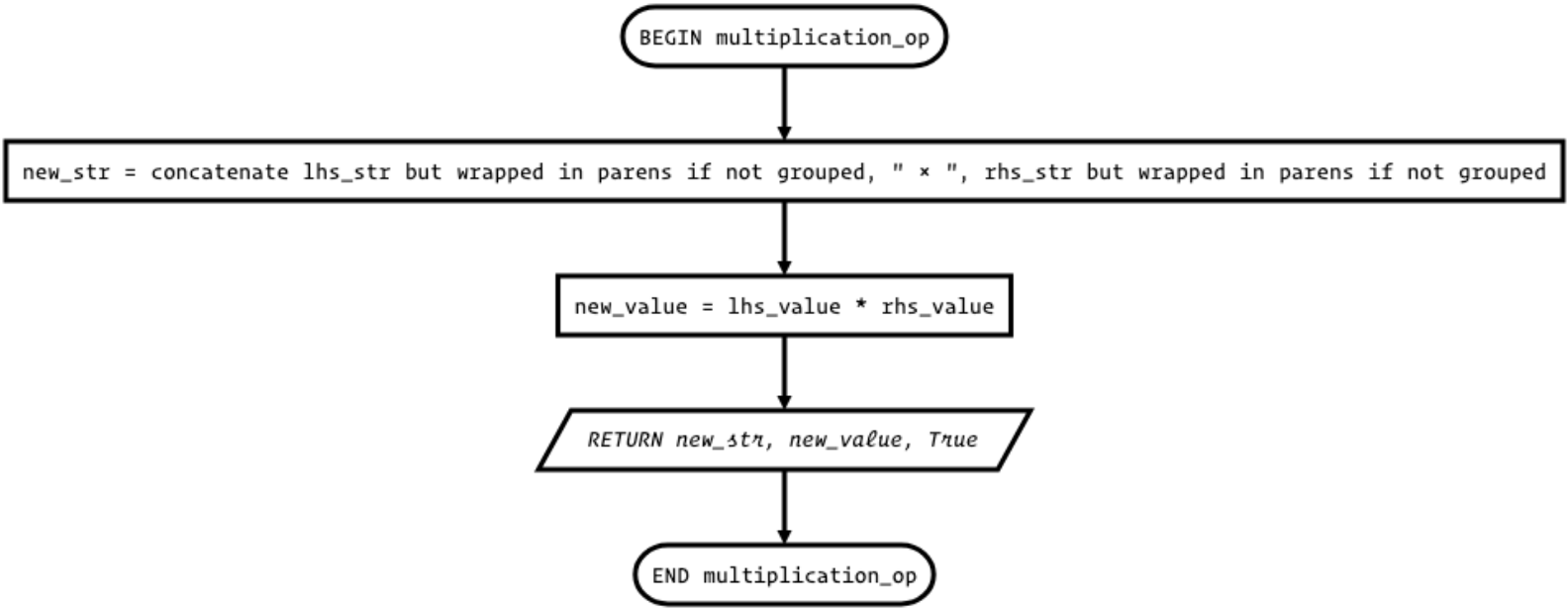

BEGIN map_range
leftSpan = leftMax - leftMin
rightSpan = rightMax - rightMin
valueScaled = (value - leftMin) / leftSpan
result = rightMin + (valueScaled * rightSpan)
RETURN result
END map_range

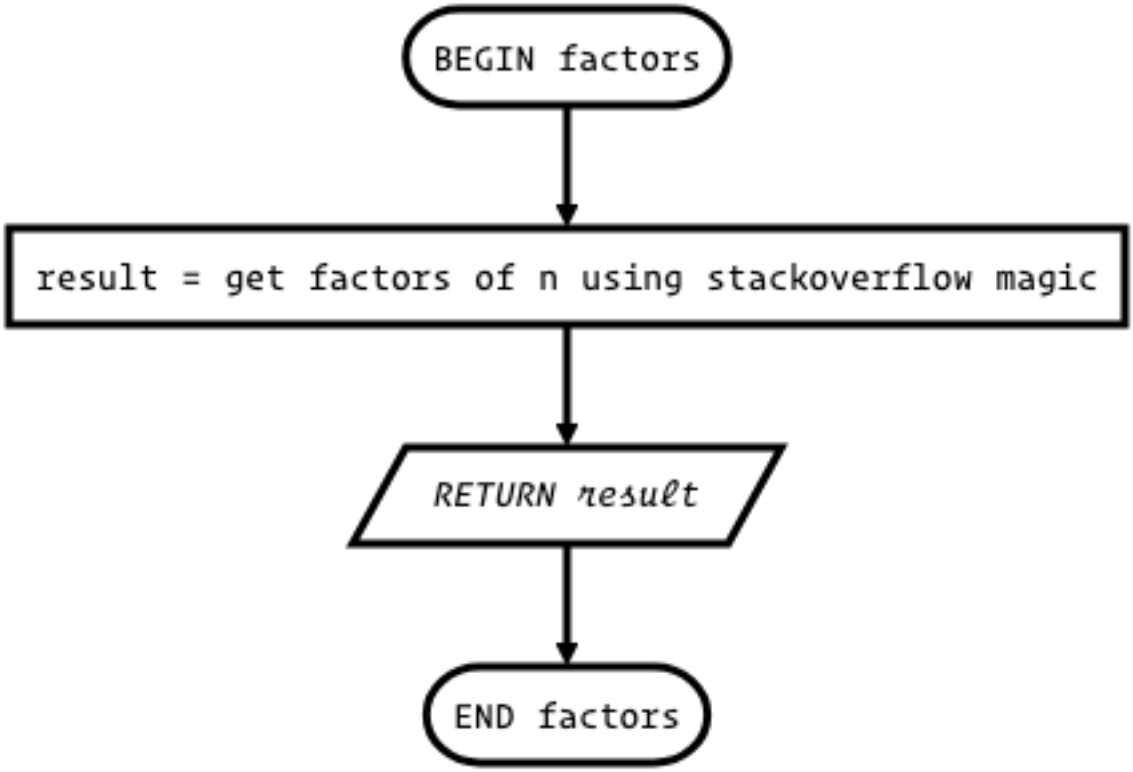

BEGIN q_bodmas
multi_ops = [addition_op, subtraction_op, multiplication_op]
single_ops = [division_op, power_op, negate_op, parens_op]
test_progress = get_test_progress(controller, question_index)
IF difficulty is normal
no
difficulty_progression_factor = round result of map_range(test_progress, 0, 1, 0, 15)
iterations = randint from (5+difficulty_progression_factor) to (10+difficulty_progression_factor)
yes
difficulty_progression_factor = round result of map_range(test_progress, 0, 1, 0, 4)
iterations = randint from (2+difficulty_progression_factor) to (4+difficulty_progression_factor)
(expr_str, expr_value, _) = get_q_expr()
question = concatenate "Evaluate ", expr_str
IF randint from 0-1 == 0
yes
is_ans_correct = lambda ans: ans == expr_value as string
component = InputQuestion(controller, question, get_integer_error_msg, is_ans_correct, expr_value as string)
RETURN component
no
fake_answer_range = (expr_value but positive) + 15
fake_answers = []
fake_answer = randint from -fake_answer_range to fake_answer_range
IF fake_answer == expr_value or fake_answer in fake_answers
yes
no
add fake_answer to fake_answers
IF fake_answers length == 3
no
yes
choices = list with str(expr_value) concatenated with fake_answers
shuffle choices
correct_choice_index = get index of str(expr_value) in choices
component = MultipleChoiceQuestion(controller, question, choices, correct_choice_index)
RETURN component
END q_bodmas

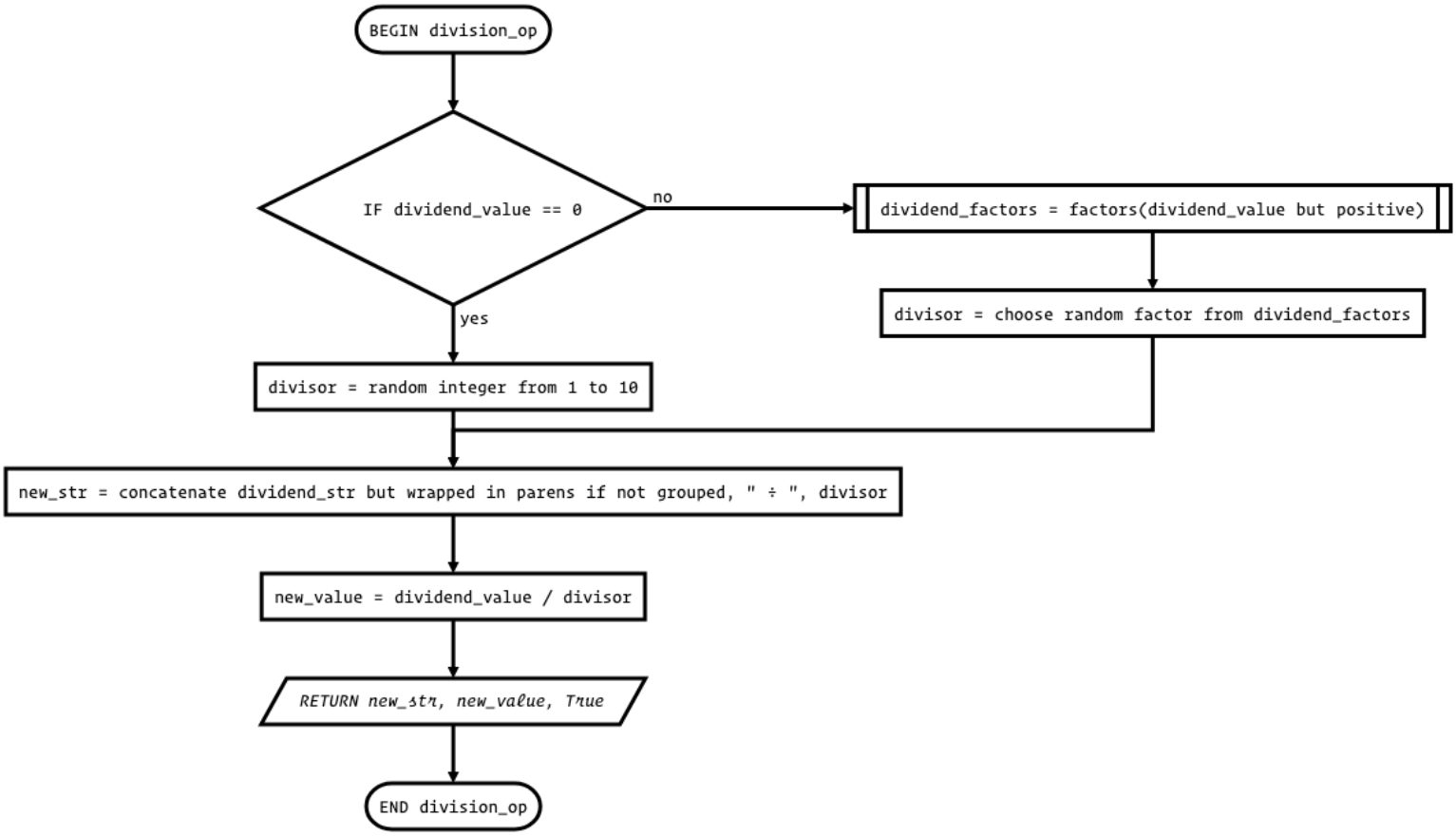

BEGIN add_parens
result = "(" + string + ")"
RETURN result
END add_parens

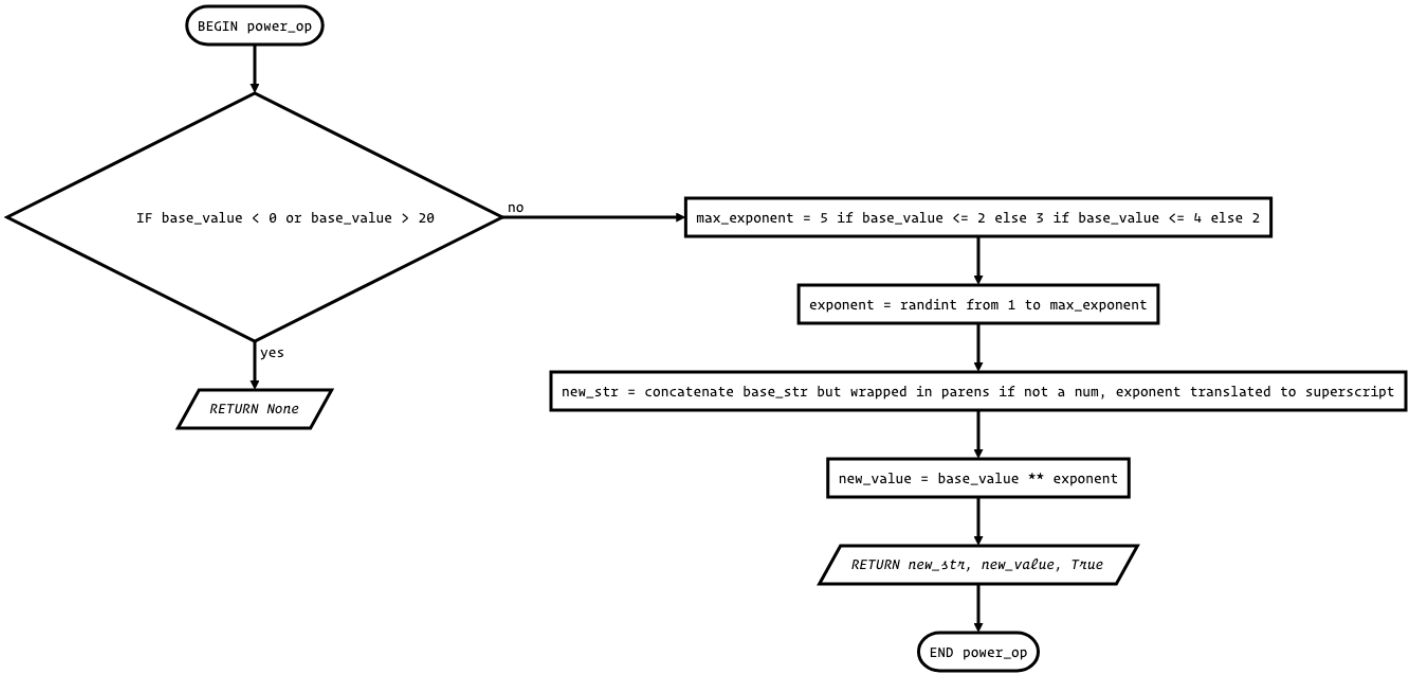

BEGIN addition_op
new_str = concatenate lhs_str, " + ", rhs_str but wrapped in parens if begins with "-"
new_value = lhs_value + rhs_value
RETURN new_str, new_value, False
END addition_op

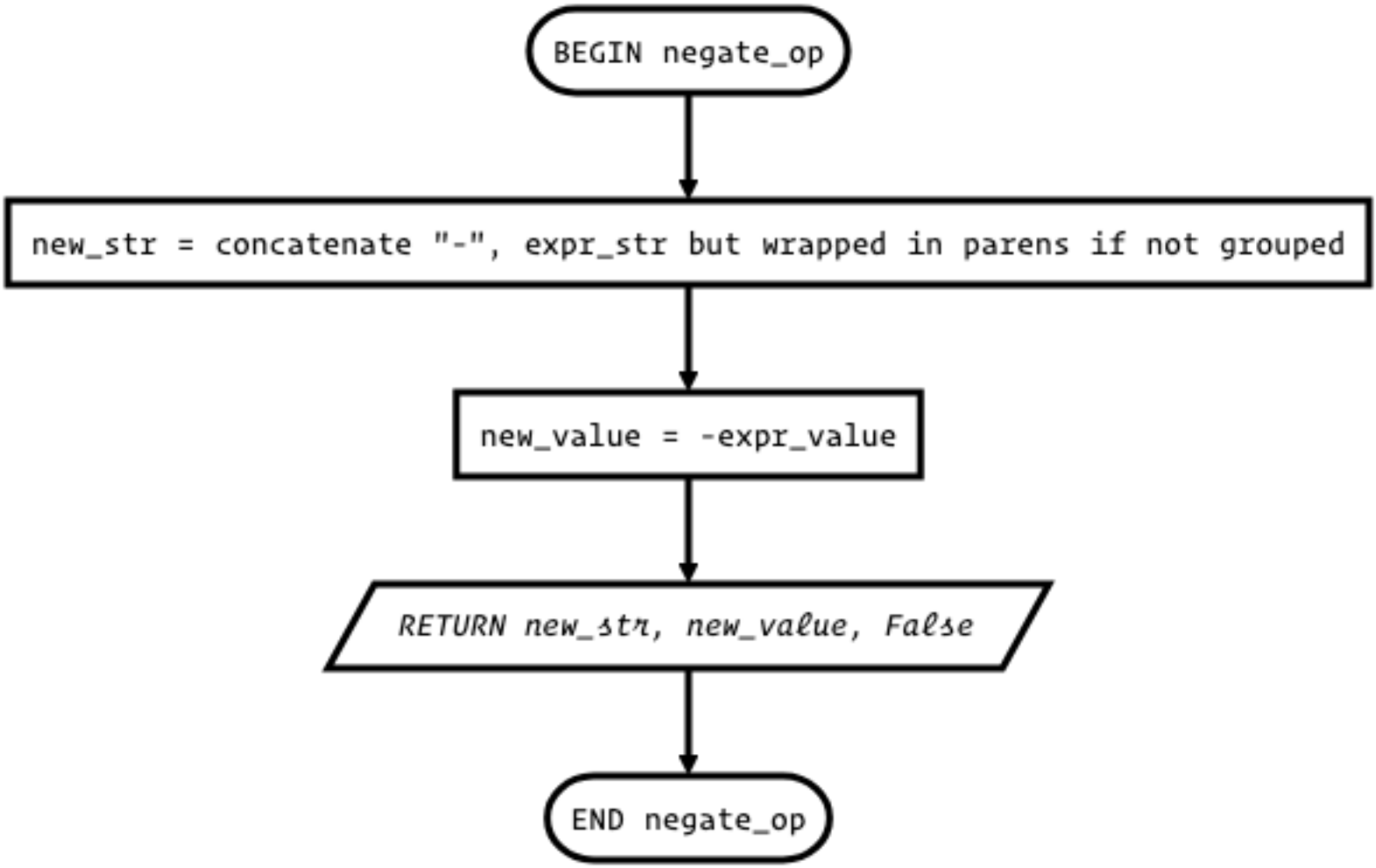

BEGIN subtraction_op
new_str = concatenate lhs_str, " - ", rhs_str but wrapped in parens if not grouped
new_value = lhs_value - rhs_value
RETURN new_str, new_value, False
END subtraction_op

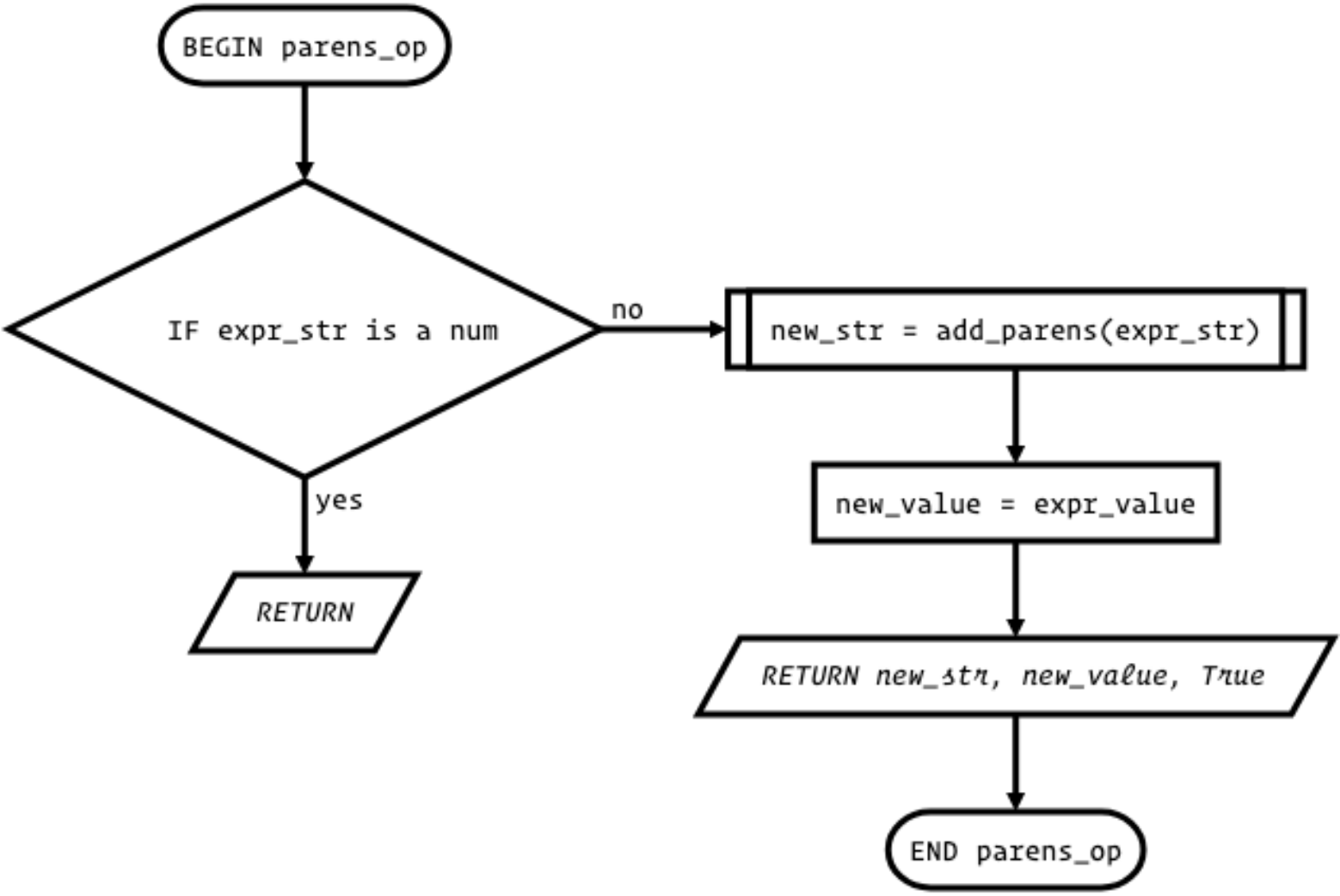

BEGIN multiplication_op
new_str = concatenate lhs_str but wrapped in parens if not grouped, " × ", rhs_str but wrapped in parens if not grouped
new_value = lhs_value * rhs_value
RETURN new_str, new_value, True
END multiplication_op

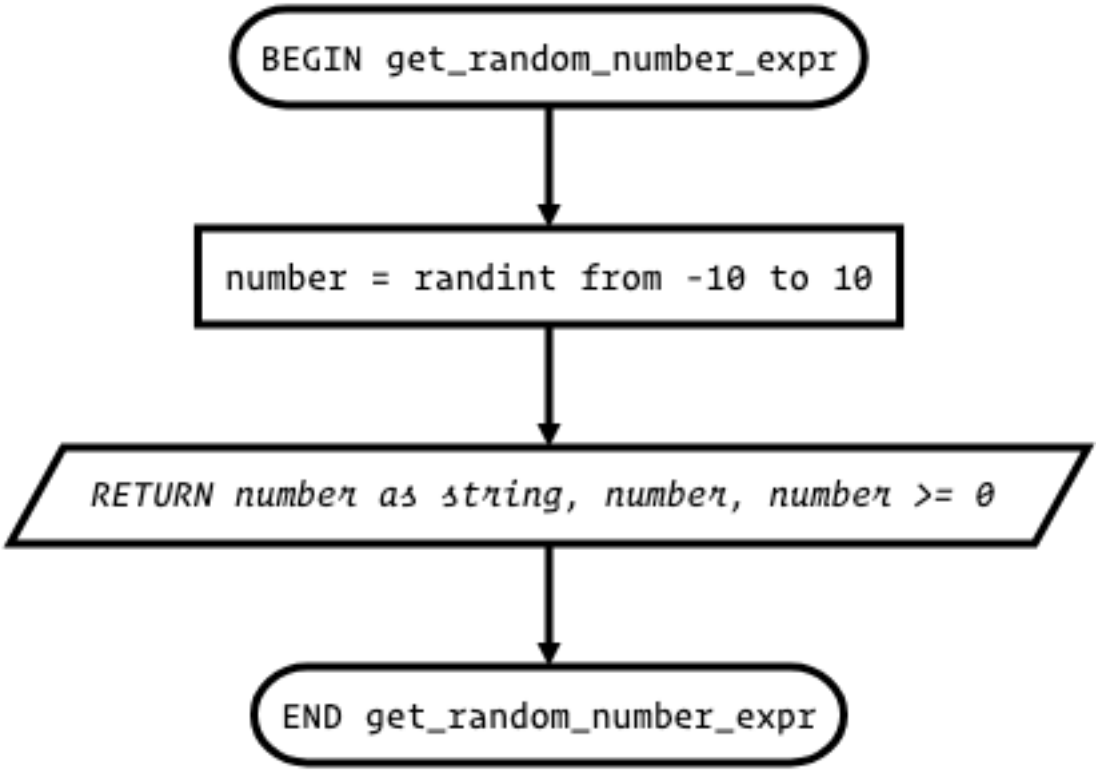

BEGIN factors
result = get factors of n using stackoverflow magic
RETURN result
END factors

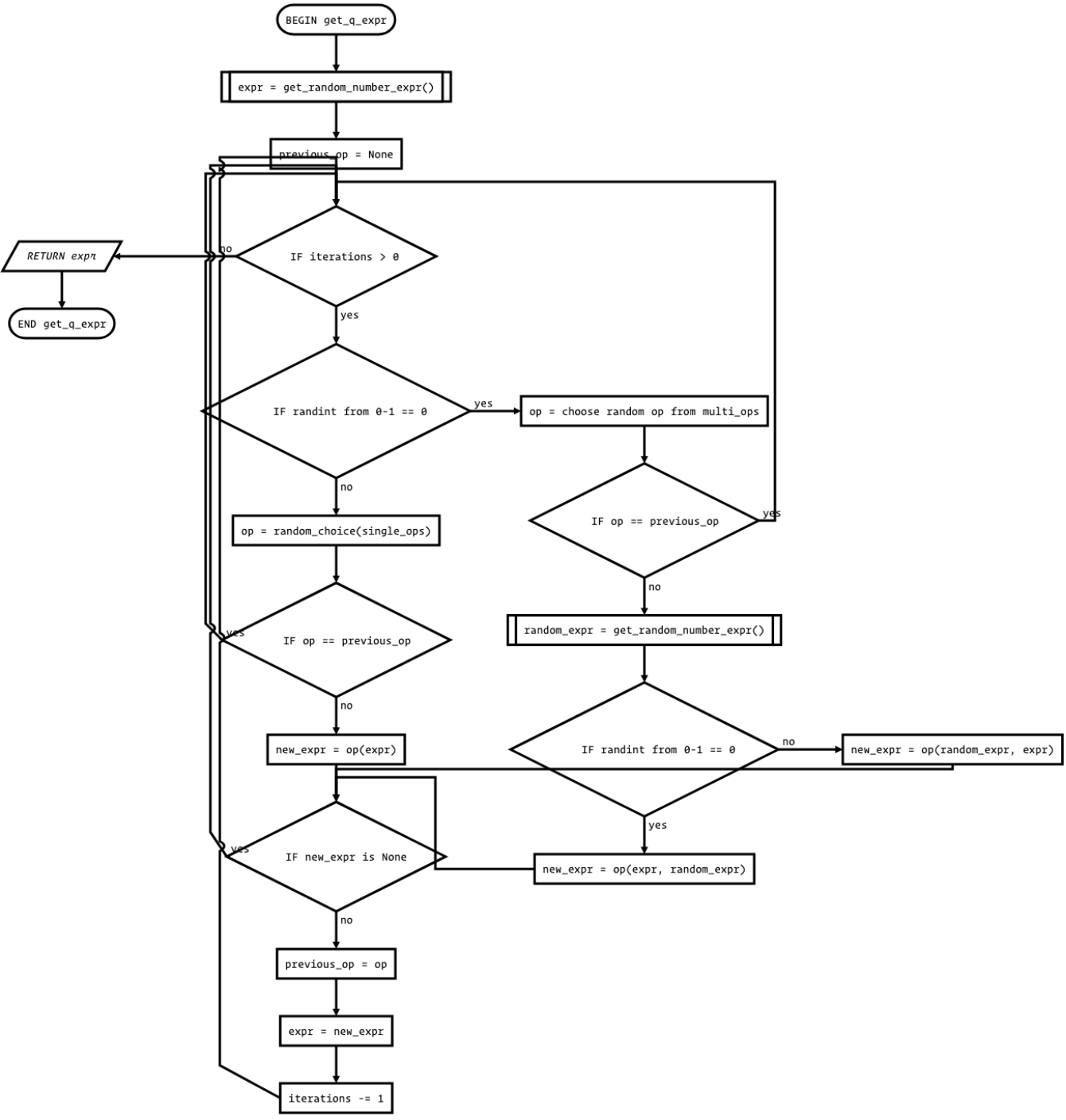

BEGIN division_op
IF dividend_value == 0
no
yes
dividend_factors = factors(dividend_value but positive)
divisor = choose random factor from dividend_factors
divisor = random integer from 1 to 10
new_str = concatenate dividend_str but wrapped in parens if not grouped, " ÷ ", divisor
new_value = dividend_value / divisor
RETURN new_str, new_value, True
END division_op

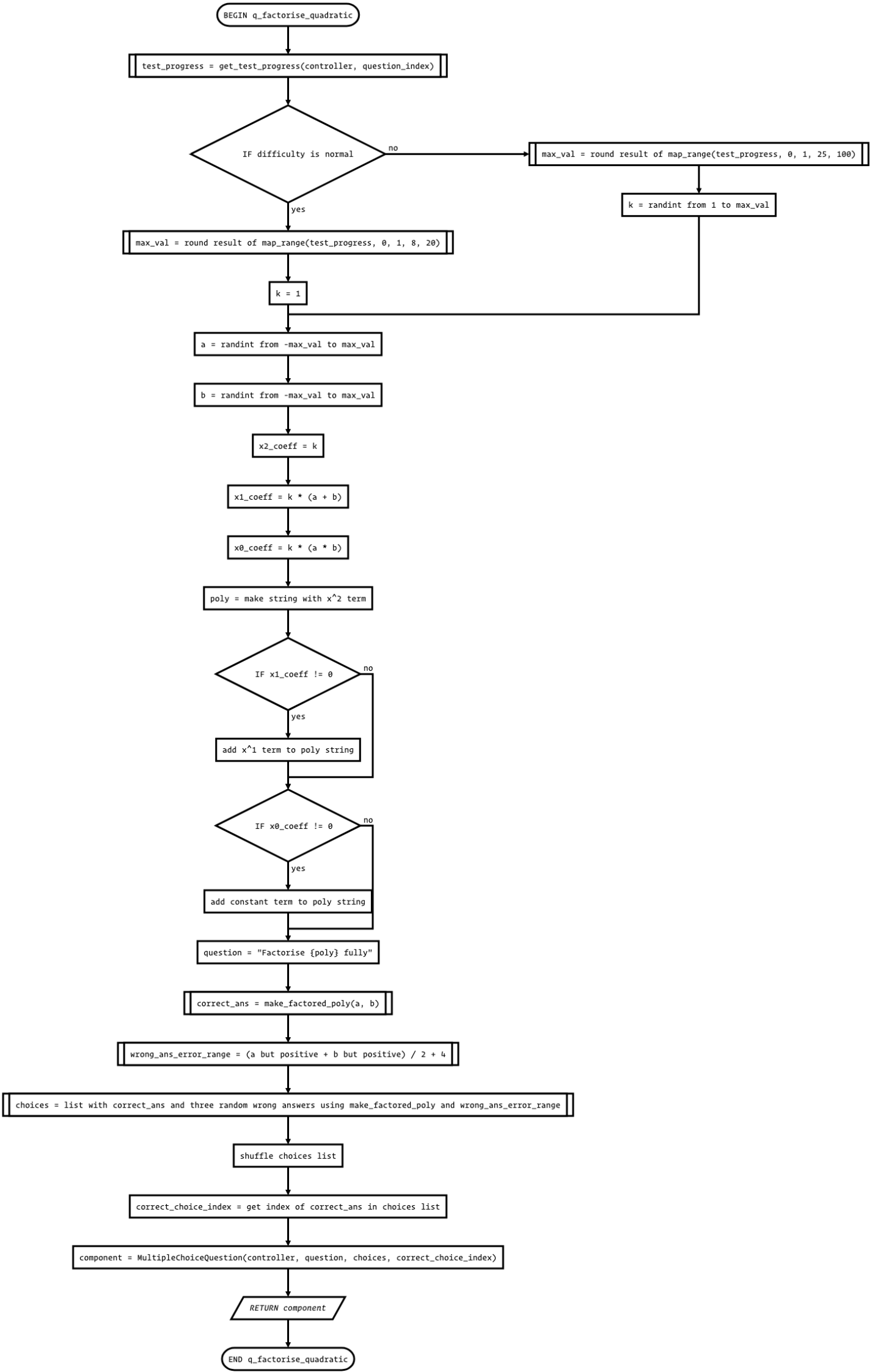

BEGIN power_op
IF base_value < 0 or base_value > 20
no
yes
RETURN None
max_exponent = 5 if base_value <= 2 else 3 if base_value <= 4 else 2
exponent = randint from 1 to max_exponent
new_str = concatenate base_str but wrapped in parens if not a num, exponent translated to superscript
new_value = base_value ** exponent
RETURN new_str, new_value, True
END power_op

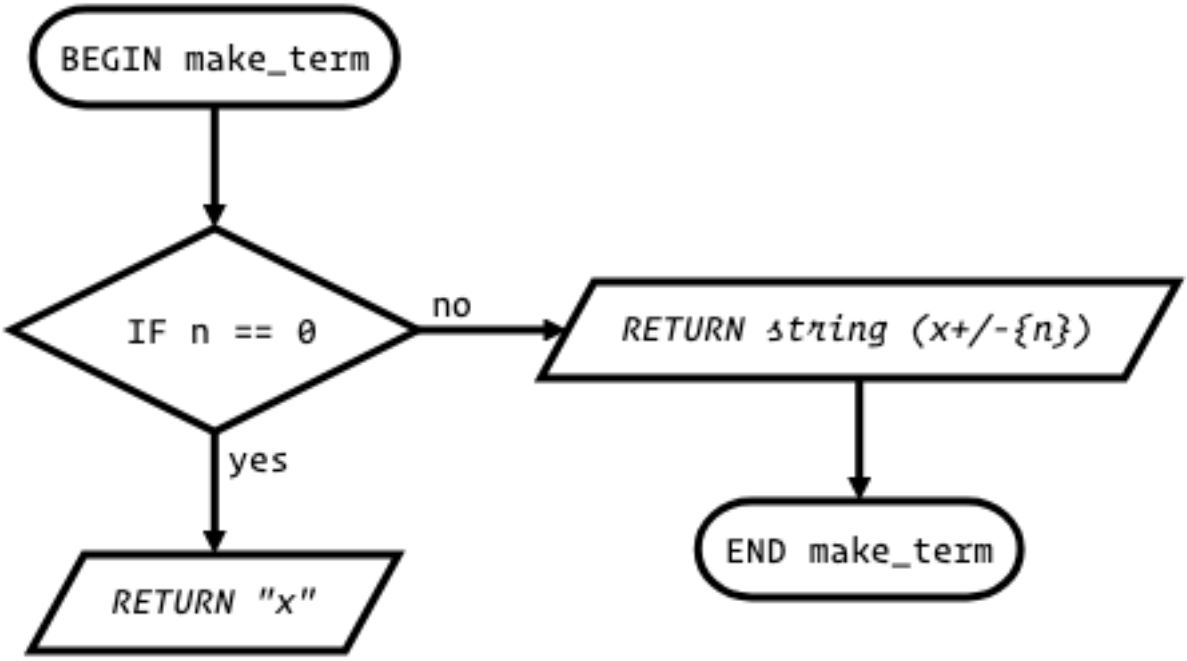

BEGIN negate_op
new_str = concatenate "-", expr_str but wrapped in parens if not grouped
new_value = -expr_value
RETURN new_str, new_value, False
END negate_op

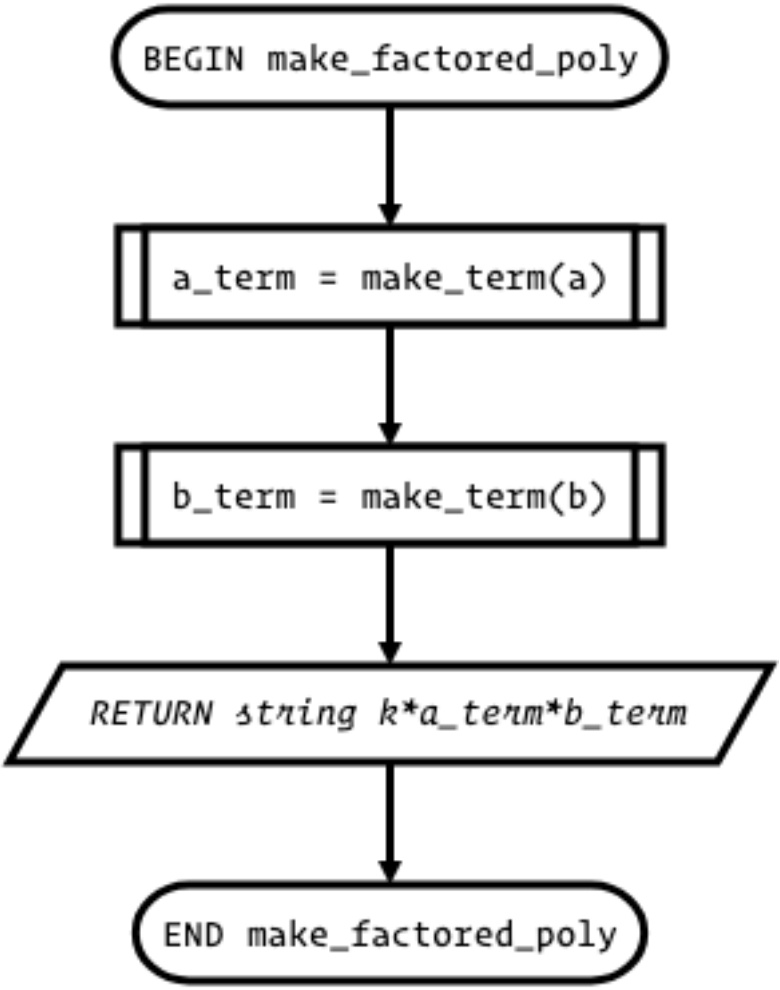

BEGIN parens_op
IF expr_str is a num
no
yes
new_str = add_parens(expr_str)
new_value = expr_value
RETURN
RETURN new_str, new_value, True
END parens_op

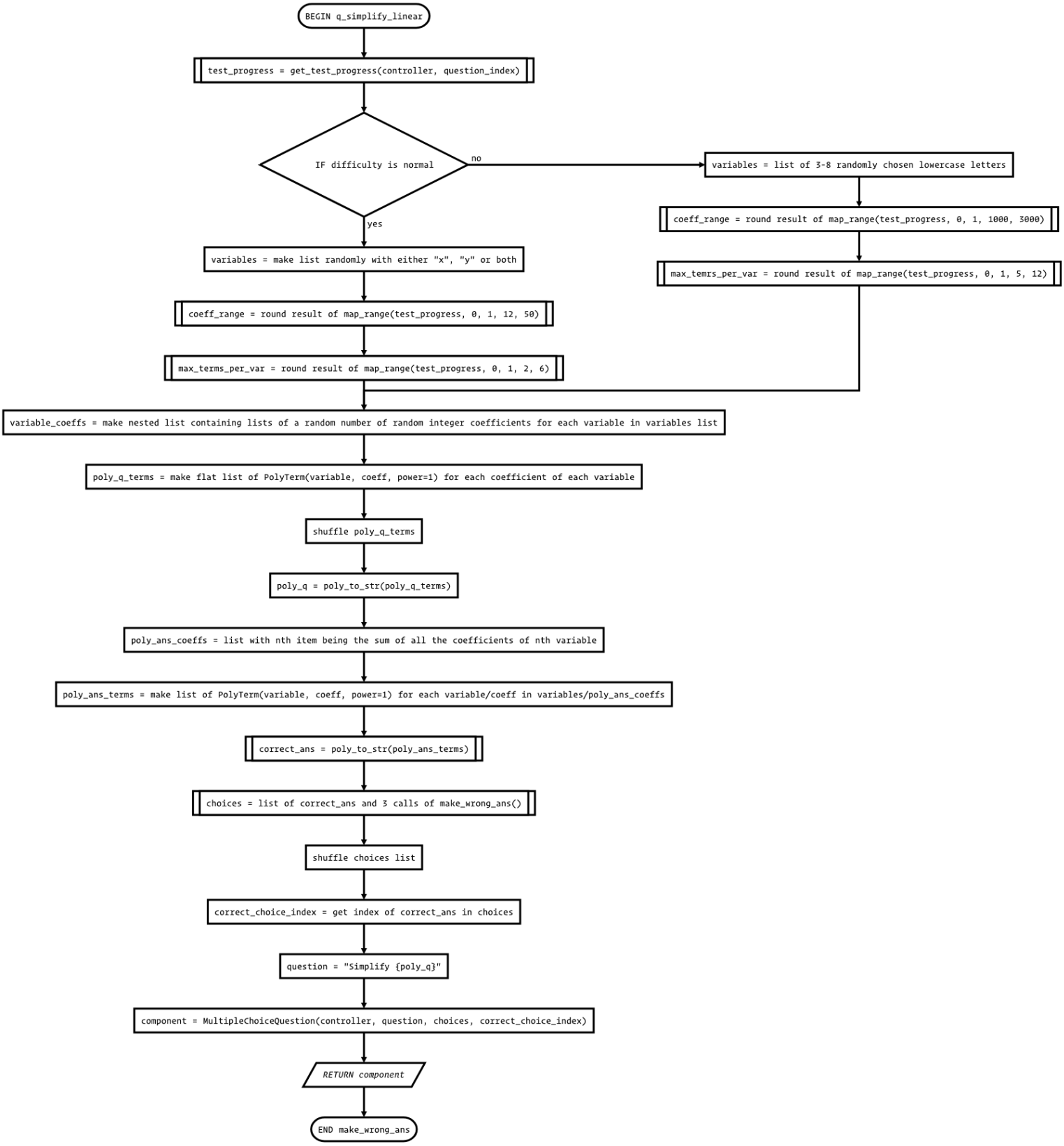

BEGIN get_random_number_expr
number = randint from -10 to 10
RETURN number as string, number, number >= 0
END get_random_number_expr

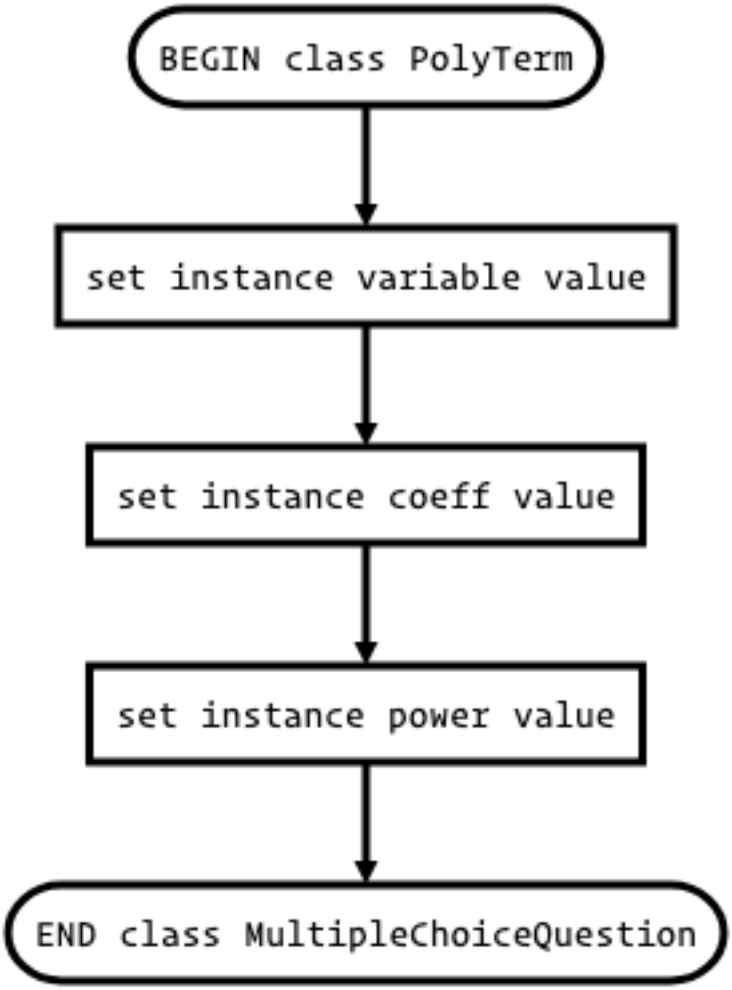

BEGIN get_q_expr
expr = get_random_number_expr()
previous_op = None
IF iterations > 0
no
RETURN expr
END get_q_expr
yes
IF randint from 0-1 == 0
yes
op = choose random op from multi_ops
IF op == previous_op
yes
no
random_expr = get_random_number_expr()
IF randint from 0-1 == 0
no
new_expr = op(random_expr, expr)
yes
new_expr = op(expr, random_expr)
no
op = random_choice(single_ops)
IF op == previous_op
yes
no
new_expr = op(expr)
IF new_expr is None
yes
no
previous_op = op
expr = new_expr
iterations -= 1

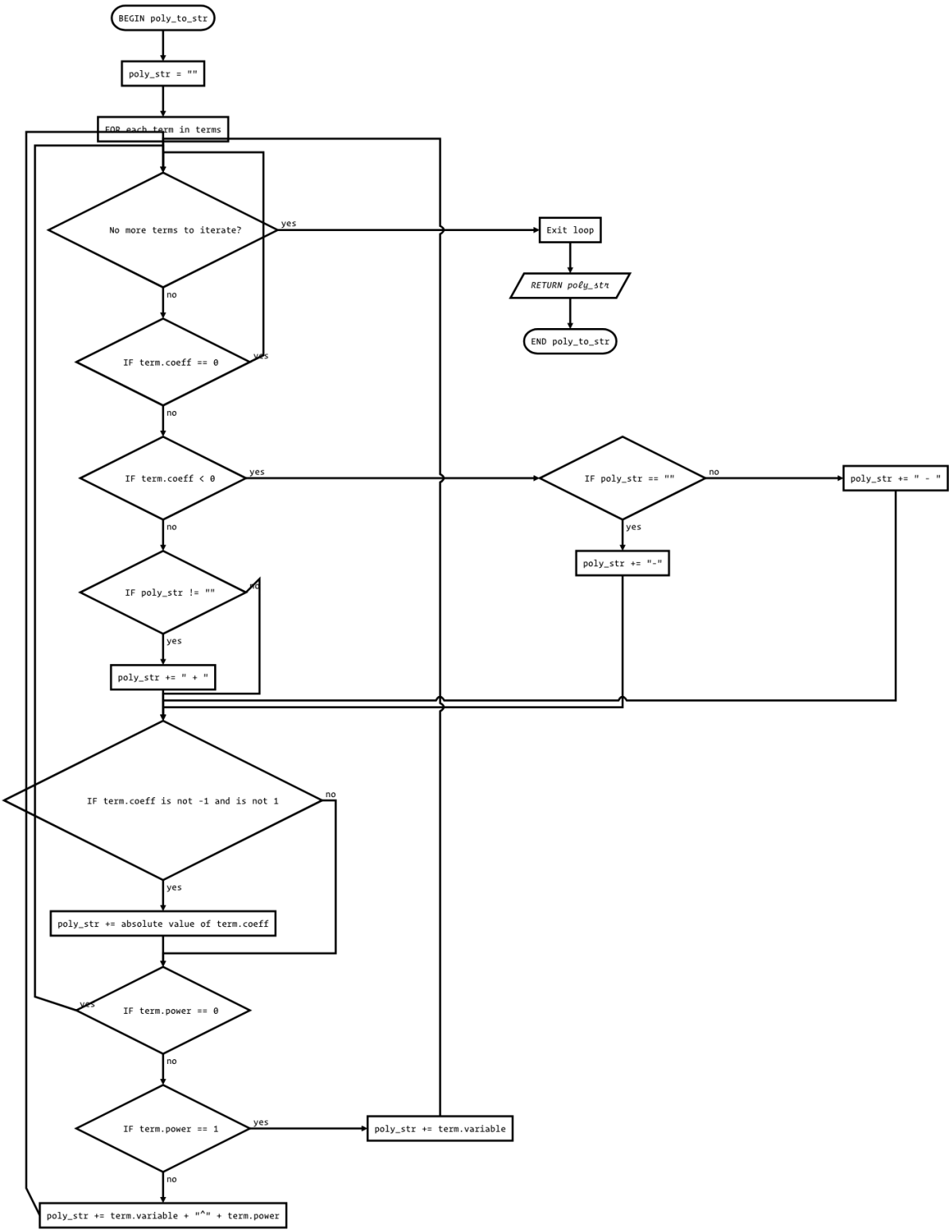

BEGIN q_factorise_quadratic
test_progress = get_test_progress(controller, question_index)
IF difficulty is normal
no
max_val = round result of map_range(test_progress, 0, 1, 25, 100)
k = randint from 1 to max_val
yes
max_val = round result of map_range(test_progress, 0, 1, 8, 20)
k = 1
a = randint from -max_val to max_val
b = randint from -max_val to max_val
x2_coeff = k
x1_coeff = k * (a + b)
x0_coeff = k * (a * b)
poly = make string with x^2 term
IF x1_coeff != 0
no
yes
add x^1 term to poly string
IF x0_coeff != 0
no
yes
add constant term to poly string
question = "Factorise {poly} fully"
correct_ans = make_factored_poly(a, b)
wrong_ans_error_range = (a but positive + b but positive) / 2 + 4
choices = list with correct_ans and three random wrong answers using make_factored_poly and wrong_ans_error_range
shuffle choices list
correct_choice_index = get index of correct_ans in choices list
component = MultipleChoiceQuestion(controller, question, choices, correct_choice_index)
RETURN component
END q_factorise_quadratic

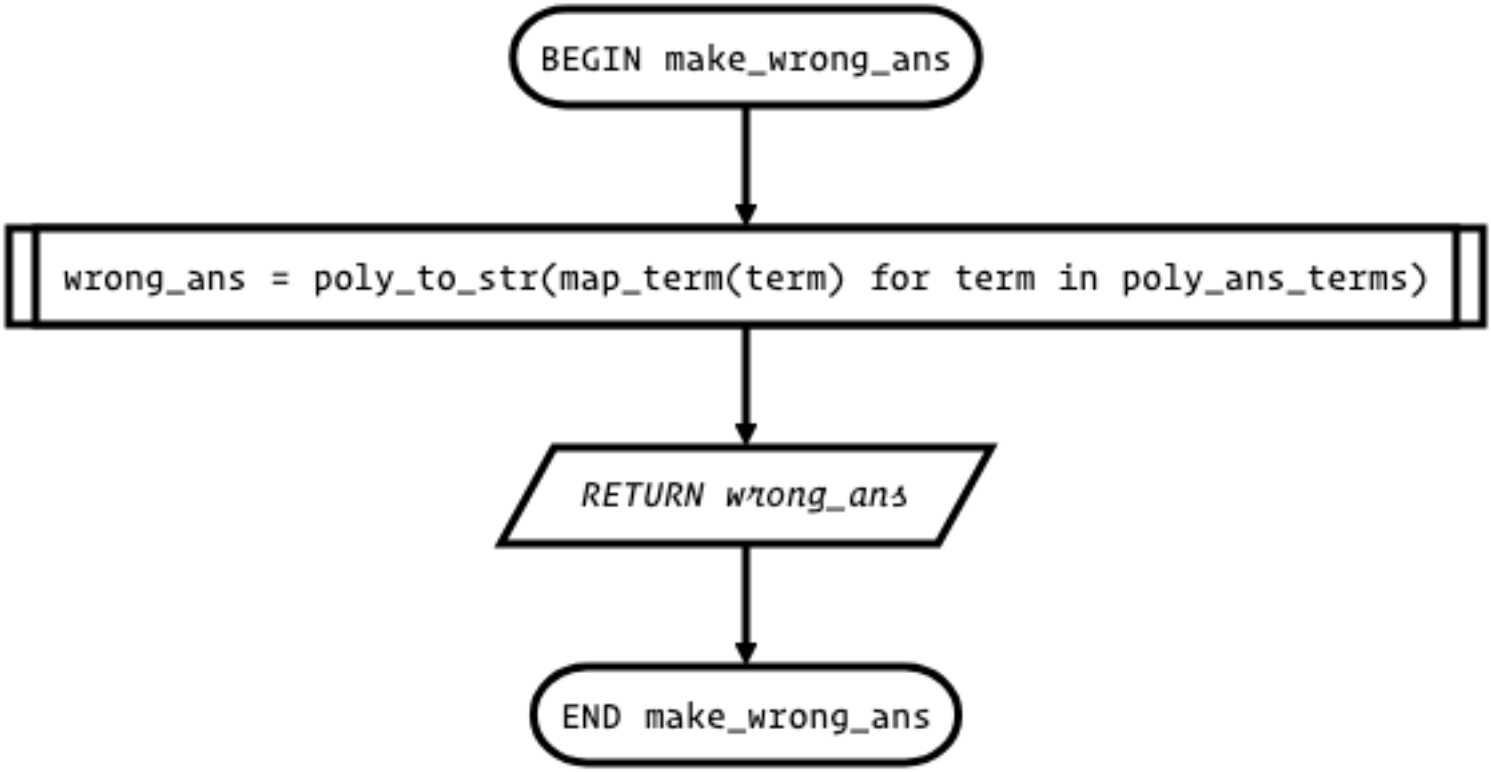

BEGIN make_term
IF n == 0
no
RETURN string (x+/-{n})
yes
END make_term
RETURN "x"

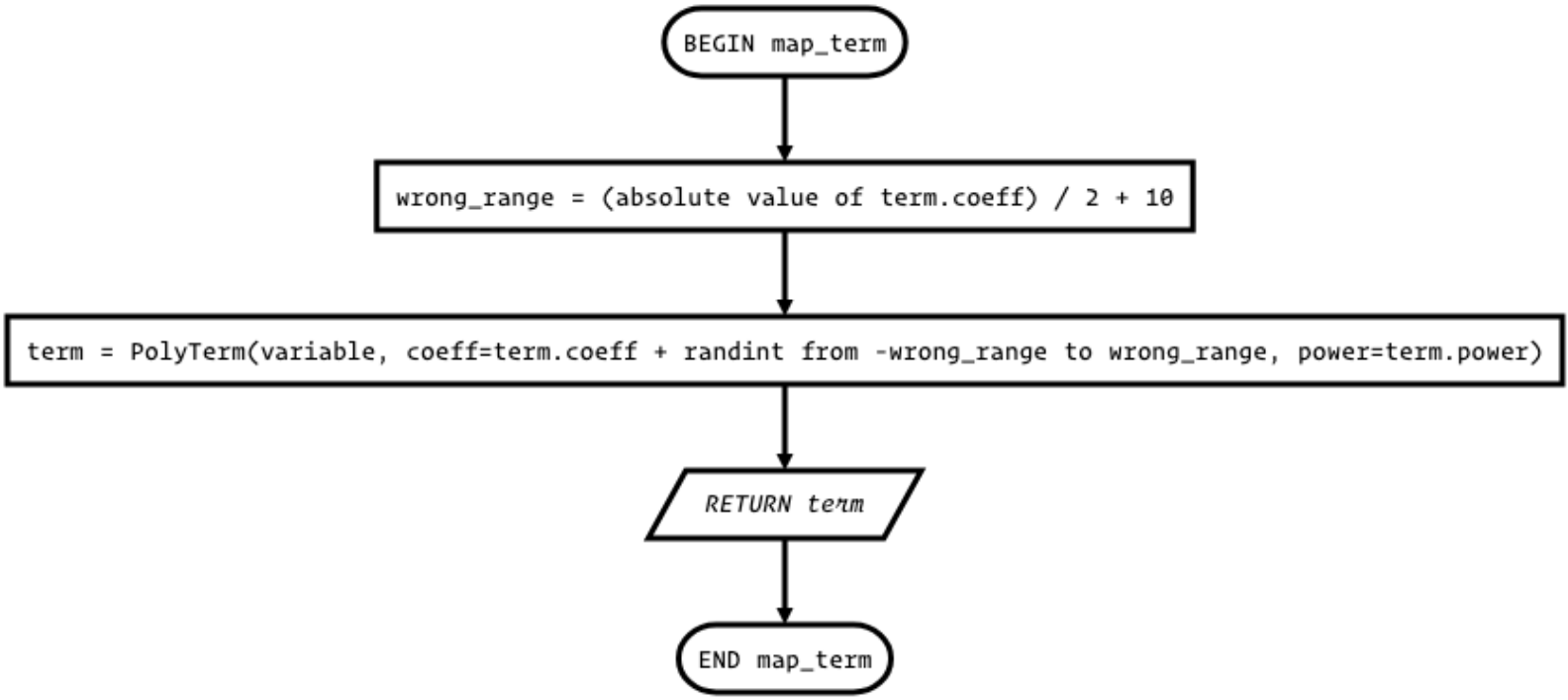

BEGIN make_factored_poly
a_term = make_term(a)
b_term = make_term(b)
RETURN string k*a_term*b_term
END make_factored_poly

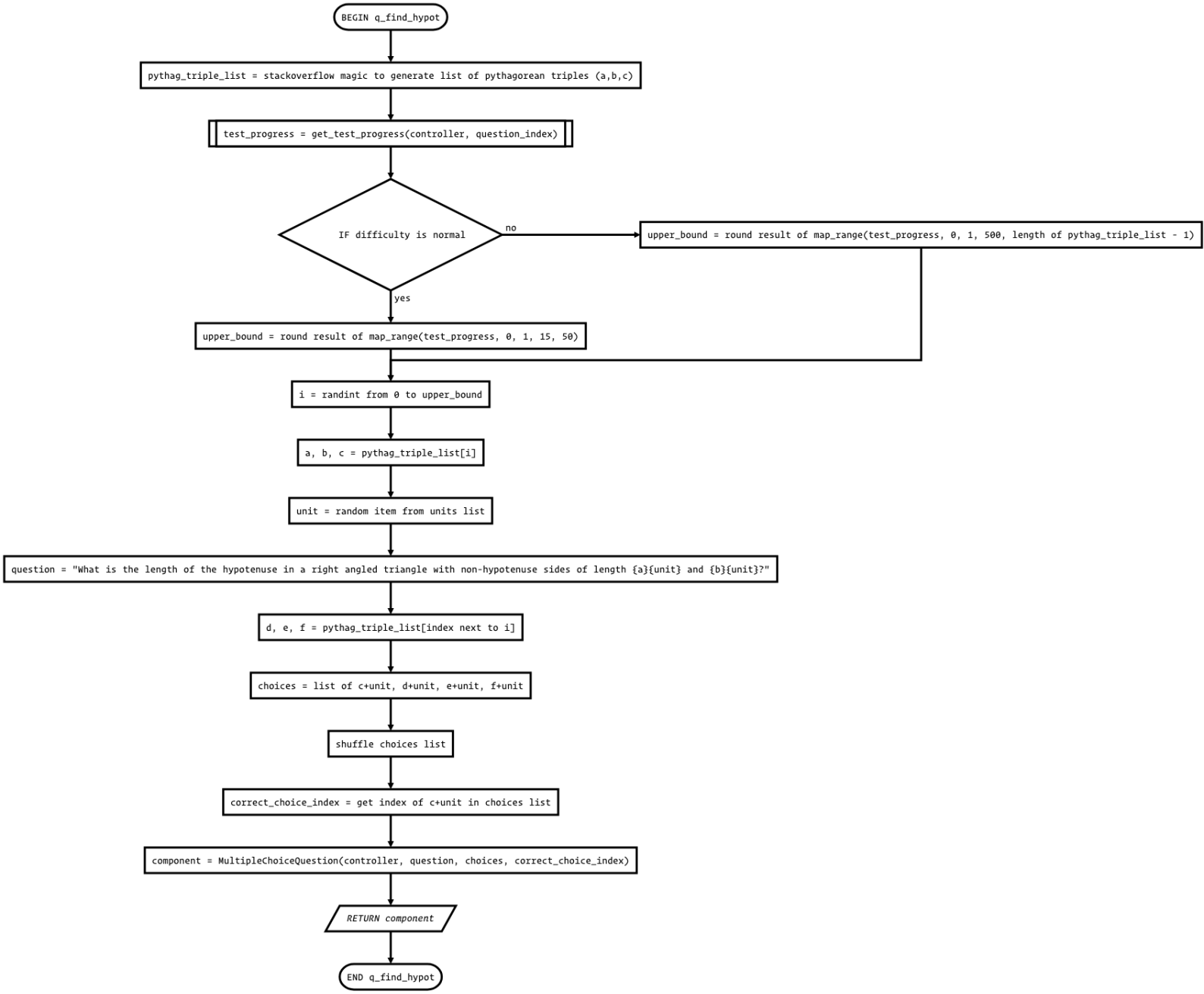

BEGIN q_simplify_linear
test_progress = get_test_progress(controller, question_index)
IF difficulty is normal
no
yes
variables = list of 3-8 randomly chosen lowercase letters
coeff_range = round result of map_range(test_progress, 0, 1, 1000, 3000)
max_temrs_per_var = round result of map_range(test_progress, 0, 1, 5, 12)
variables = make list randomly with either "x", "y" or both
coeff_range = round result of map_range(test_progress, 0, 1, 12, 50)
max_terms_per_var = round result of map_range(test_progress, 0, 1, 2, 6)
variable_coeffs = make nested list containing lists of a random number of random integer coefficients for each variable in variables list
poly_q_terms = make flat list of PolyTerm(variable, coeff, power=1) for each coefficient of each variable
shuffle poly_q_terms
poly_q = poly_to_str(poly_q_terms)
poly_ans_coeffs = list with nth item being the sum of all the coefficients of nth variable
poly_ans_terms = make list of PolyTerm(variable, coeff, power=1) for each variable/coeff in variables/poly_ans_coeffs
correct_ans = poly_to_str(poly_ans_terms)
choices = list of correct_ans and 3 calls of make_wrong_ans()
shuffle choices list
correct_choice_index = get index of correct_ans in choices
question = "Simplify {poly_q}"
component = MultipleChoiceQuestion(controller, question, choices, correct_choice_index)
RETURN component
END make_wrong_ans

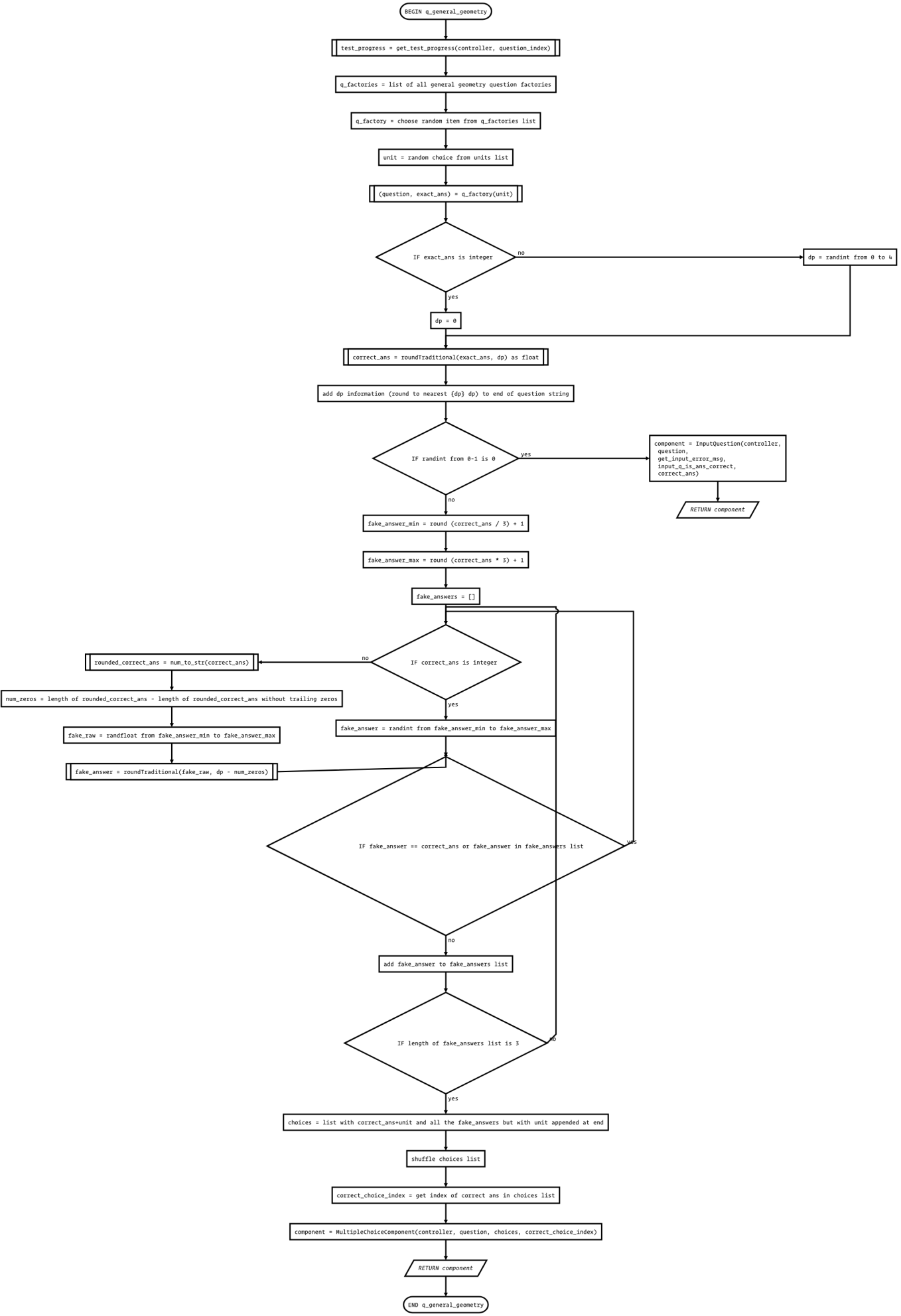

BEGIN class PolyTerm
set instance variable value
set instance coeff value
set instance power value
END class MultipleChoiceQuestion

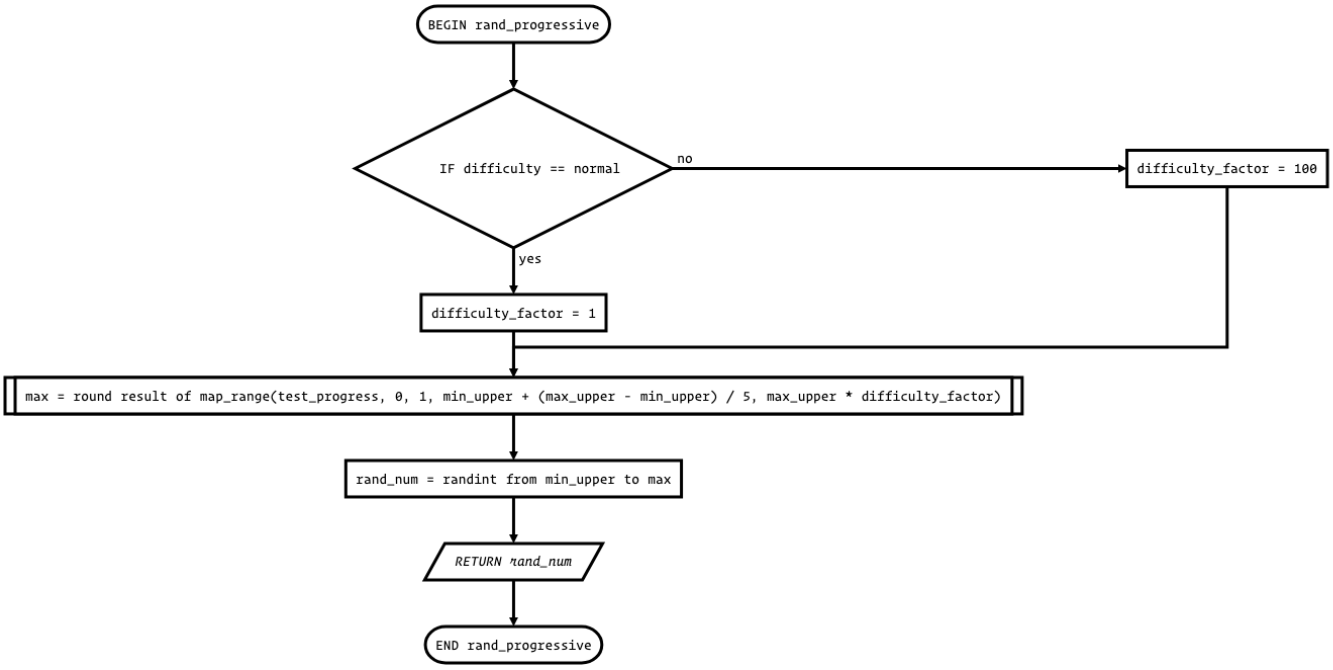

BEGIN poly_to_str
poly_str = ""
FOR each term in terms
No more terms to iterate?
yes
Exit loop
RETURN poly_str
END poly_to_str
no
IF term.coeff == 0
yes
no
IF term.coeff < 0
yes
IF poly_str == ""
no
poly_str += " - "
yes
poly_str += "-"
no
IF poly_str != ""
no
yes
poly_str += " + "
IF term.coeff is not -1 and is not 1
no
yes
poly_str += absolute value of term.coeff
IF term.power == 0
yes
no
IF term.power == 1
yes
poly_str += term.variable
no
poly_str += term.variable + "^" + term.power

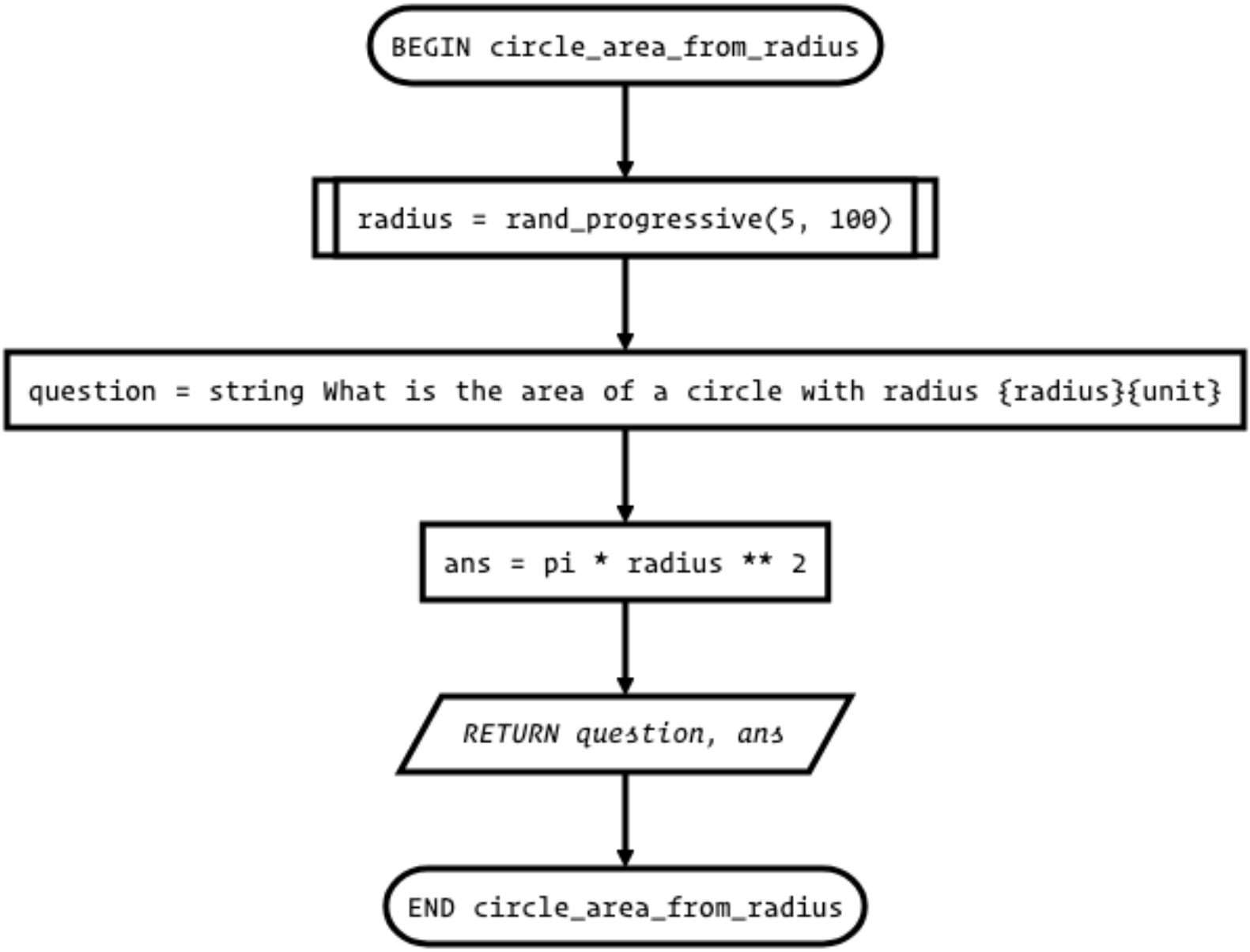

BEGIN make_wrong_ans
wrong_ans = poly_to_str(map_term(term) for term in poly_ans_terms)
RETURN wrong_ans
END make_wrong_ans

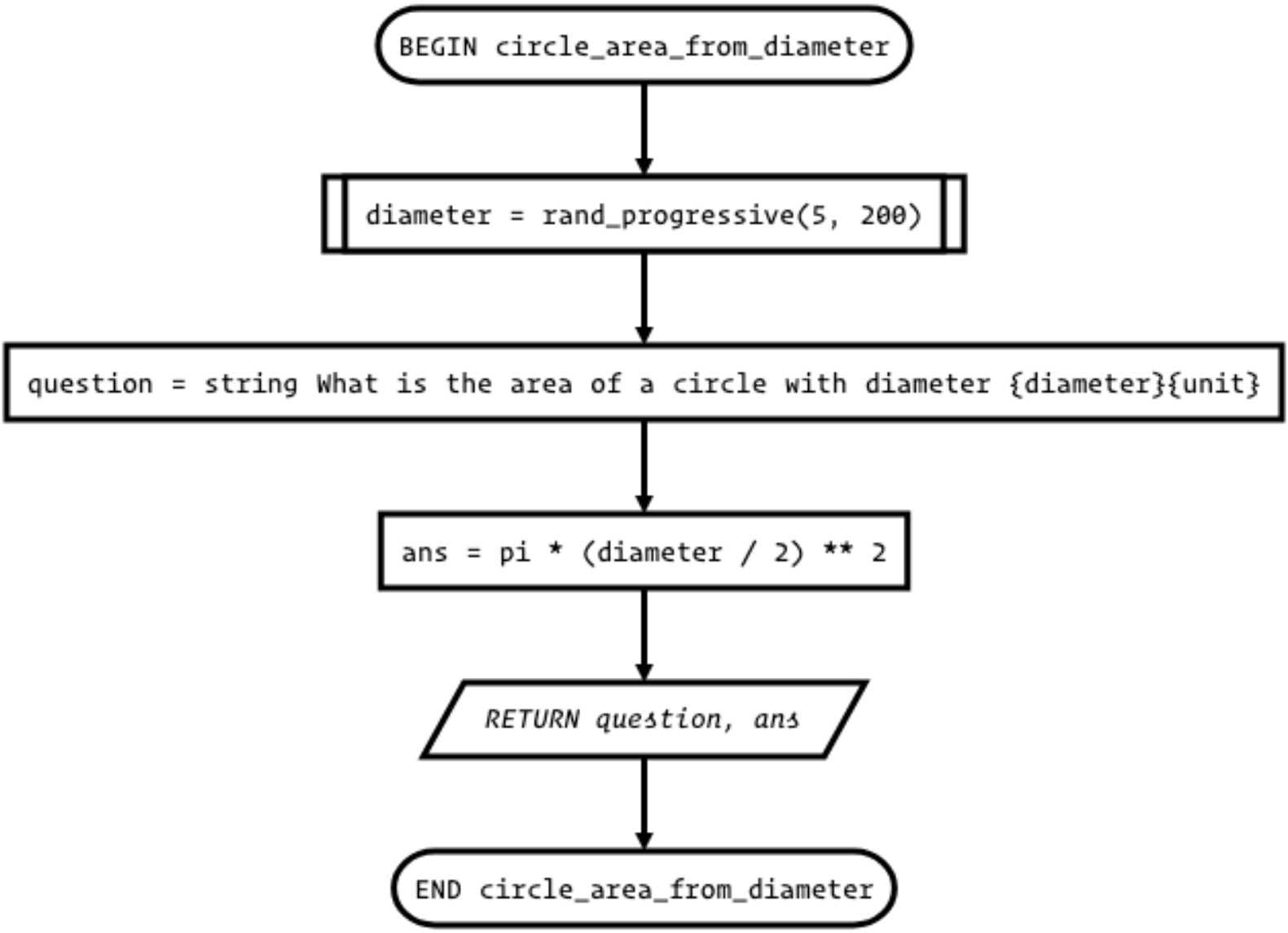

BEGIN map_term
wrong_range = (absolute value of term.coeff) / 2 + 10
term = PolyTerm(variable, coeff=term.coeff + randint from -wrong_range to wrong_range, power=term.power)
RETURN term
END map_term

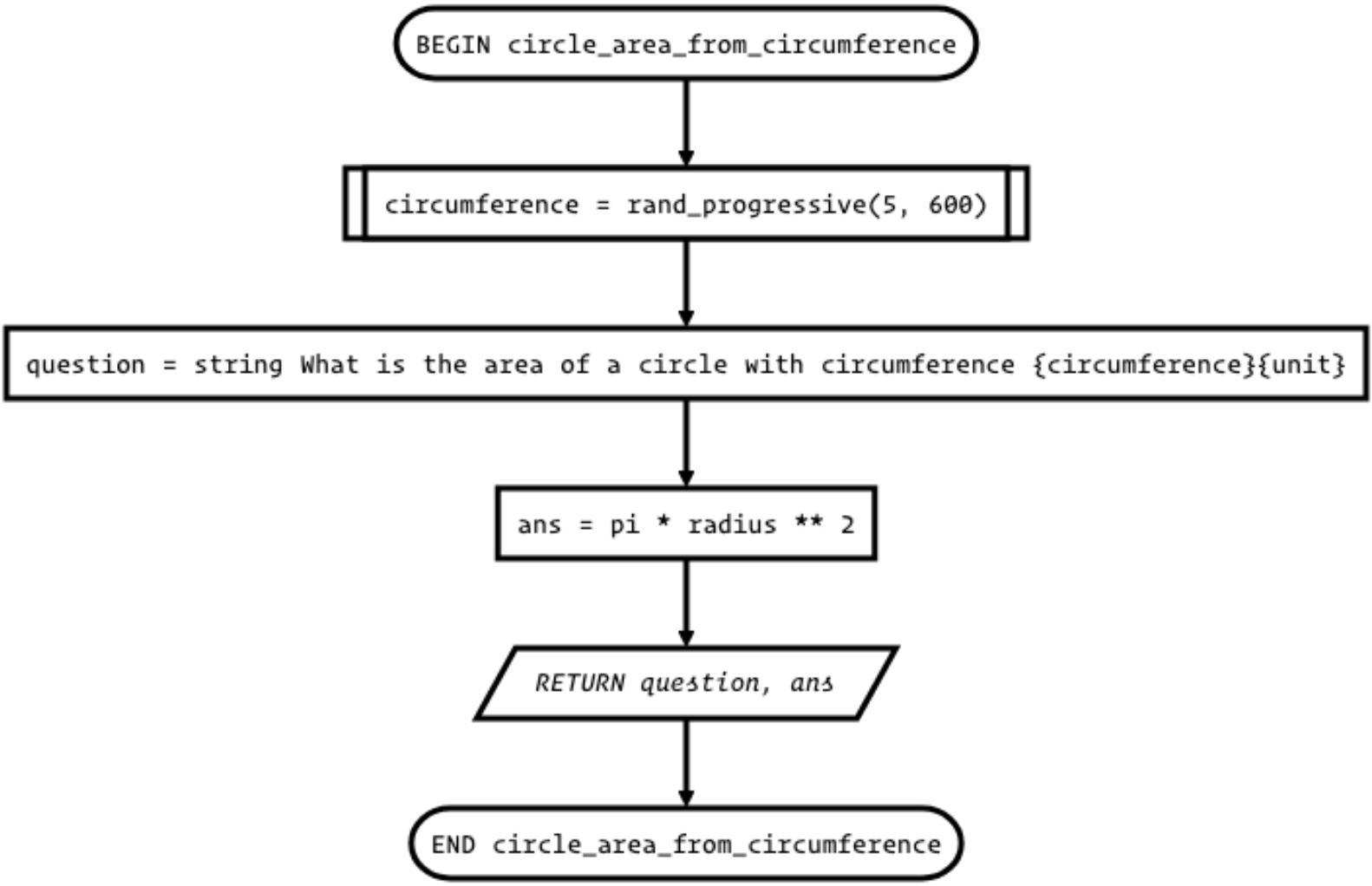

BEGIN q_find_hypot
pythag_triple_list = stackoverflow magic to generate list of pythagorean triples (a,b,c)
test_progress = get_test_progress(controller, question_index)
IF difficulty is normal
no
upper_bound = round result of map_range(test_progress, 0, 1, 500, length of pythag_triple_list - 1)
yes
upper_bound = round result of map_range(test_progress, 0, 1, 15, 50)
i = randint from 0 to upper_bound
a, b, c = pythag_triple_list[i]
unit = random item from units list
question = "What is the length of the hypotenuse in a right angled triangle with non-hypotenuse sides of length {a}{unit} and {b}{unit}?"
d, e, f = pythag_triple_list[index next to i]
choices = list of c+unit, d+unit, e+unit, f+unit
shuffle choices list
correct_choice_index = get index of c+unit in choices list
component = MultipleChoiceQuestion(controller, question, choices, correct_choice_index)
RETURN component
END q_find_hypot

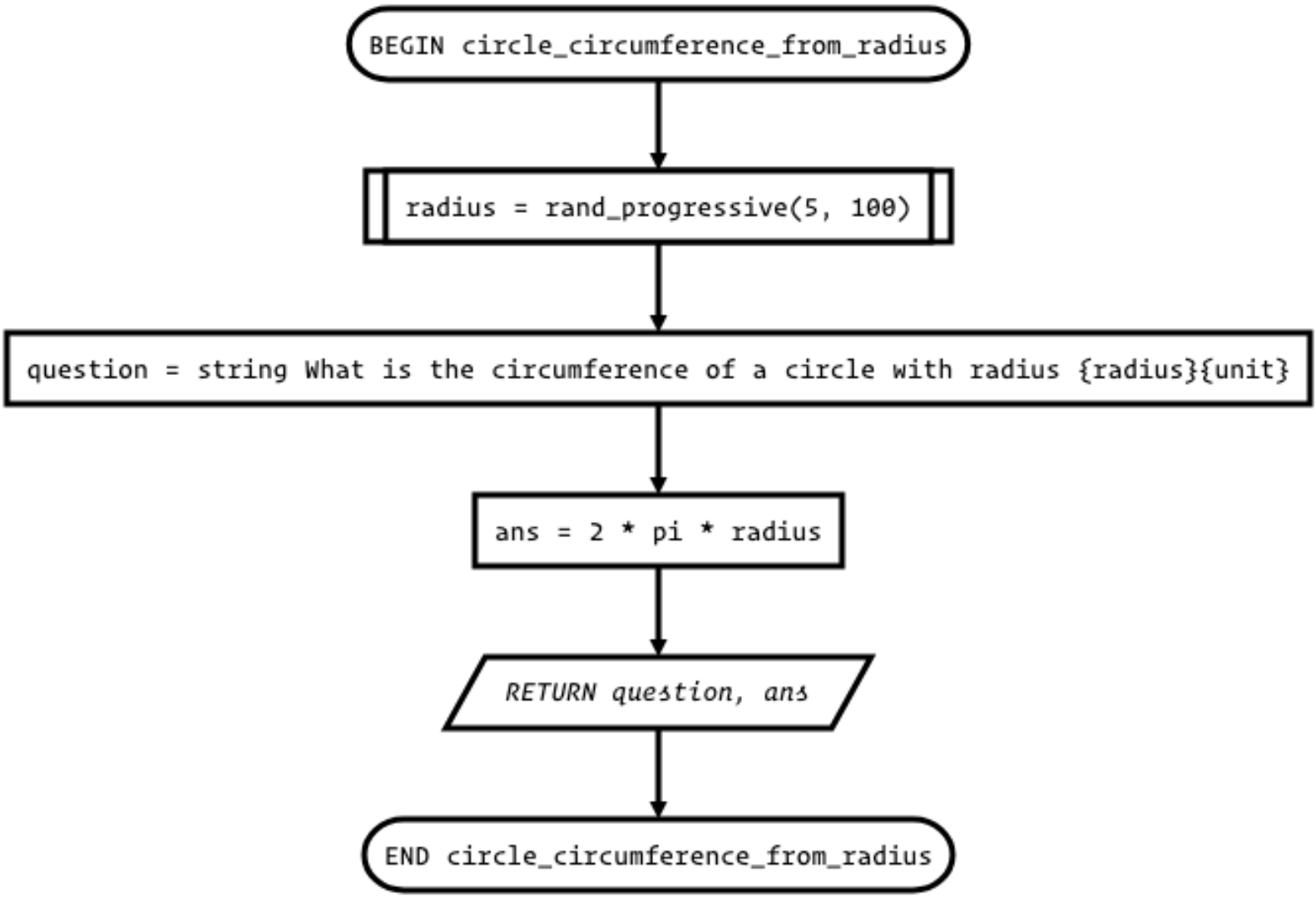

BEGIN q_general_geometry
test_progress = get_test_progress(controller, question_index)
q_factories = list of all general geometry question factories
q_factory = choose random item from q_factories list
unit = random choice from units list
(question, exact_ans) = q_factory(unit)
IF exact_ans is integer
no
dp = randint from 0 to 4
yes
dp = 0
correct_ans = roundTraditional(exact_ans, dp) as float
add dp information (round to nearest {dp} dp) to end of question string
IF randint from 0-1 is 0
yes
component = InputQuestion(controller, question, get_input_error_msg, input_q_is_ans_correct, correct_ans)
RETURN component
no
fake_answer_min = round (correct_ans / 3) + 1
fake_answer_max = round (correct_ans * 3) + 1
fake_answers = []
IF correct_ans is integer
no
rounded_correct_ans = num_to_str(correct_ans)
num_zeros = length of rounded_correct_ans - length of rounded_correct_ans without trailing zeros
fake_raw = randfloat from fake_answer_min to fake_answer_max
fake_answer = roundTraditional(fake_raw, dp - num_zeros)
yes
fake_answer = randint from fake_answer_min to fake_answer_max
IF fake_answer == correct_ans or fake_answer in fake_answers list
yes
no
add fake_answer to fake_answers list
IF length of fake_answers list is 3
no
yes
choices = list with correct_ans+unit and all the fake_answers but with unit appended at end
shuffle choices list
correct_choice_index = get index of correct ans in choices list
component = MultipleChoiceComponent(controller, question, choices, correct_choice_index)
RETURN component
END q_general_geometry

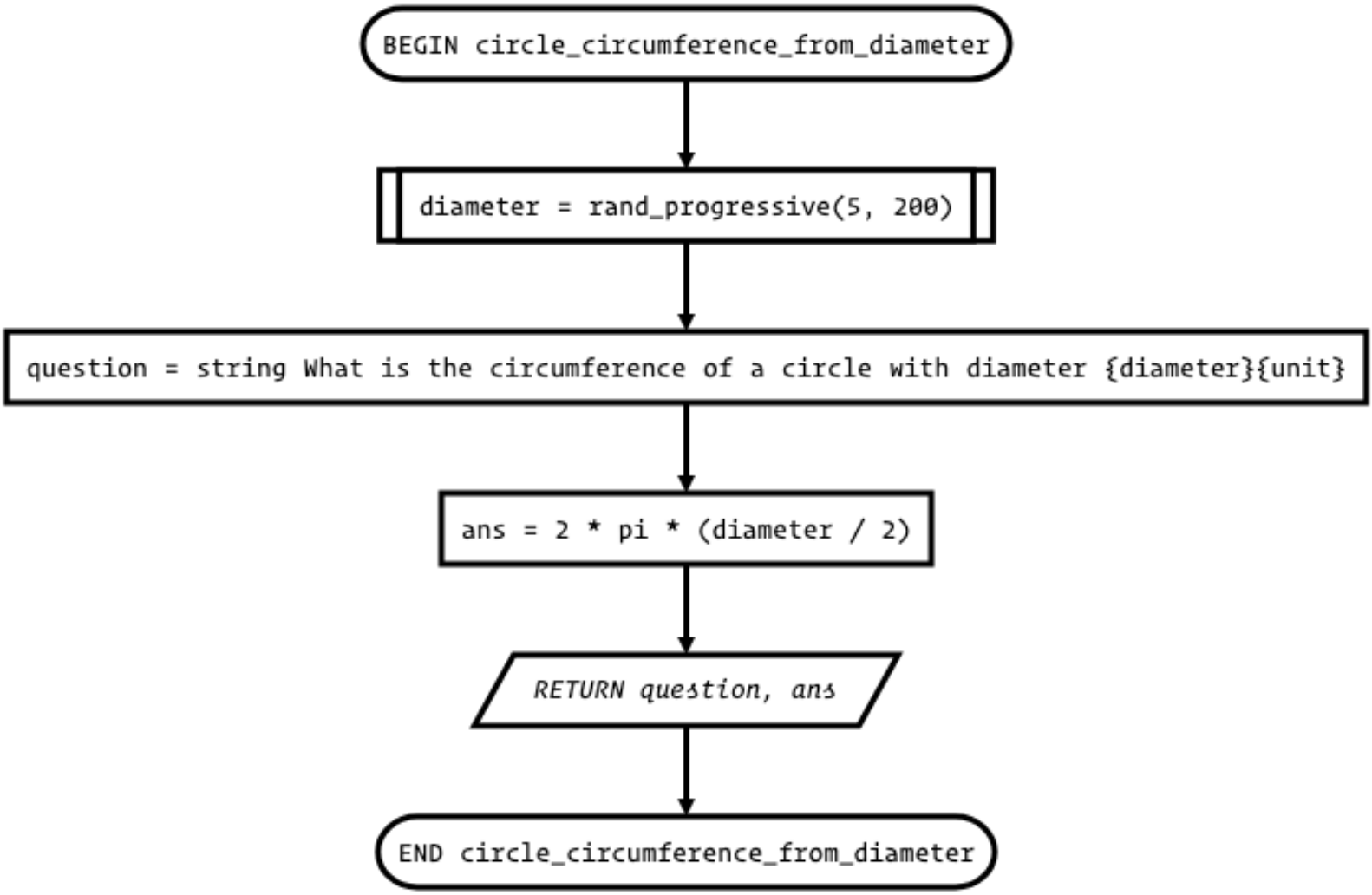

BEGIN rand_progressive
IF difficulty == normal
no
difficulty_factor = 100
yes
difficulty_factor = 1
max = round result of map_range(test_progress, 0, 1, min_upper + (max_upper - min_upper) / 5, max_upper * difficulty_factor)
rand_num = randint from min_upper to max
RETURN rand_num
END rand_progressive

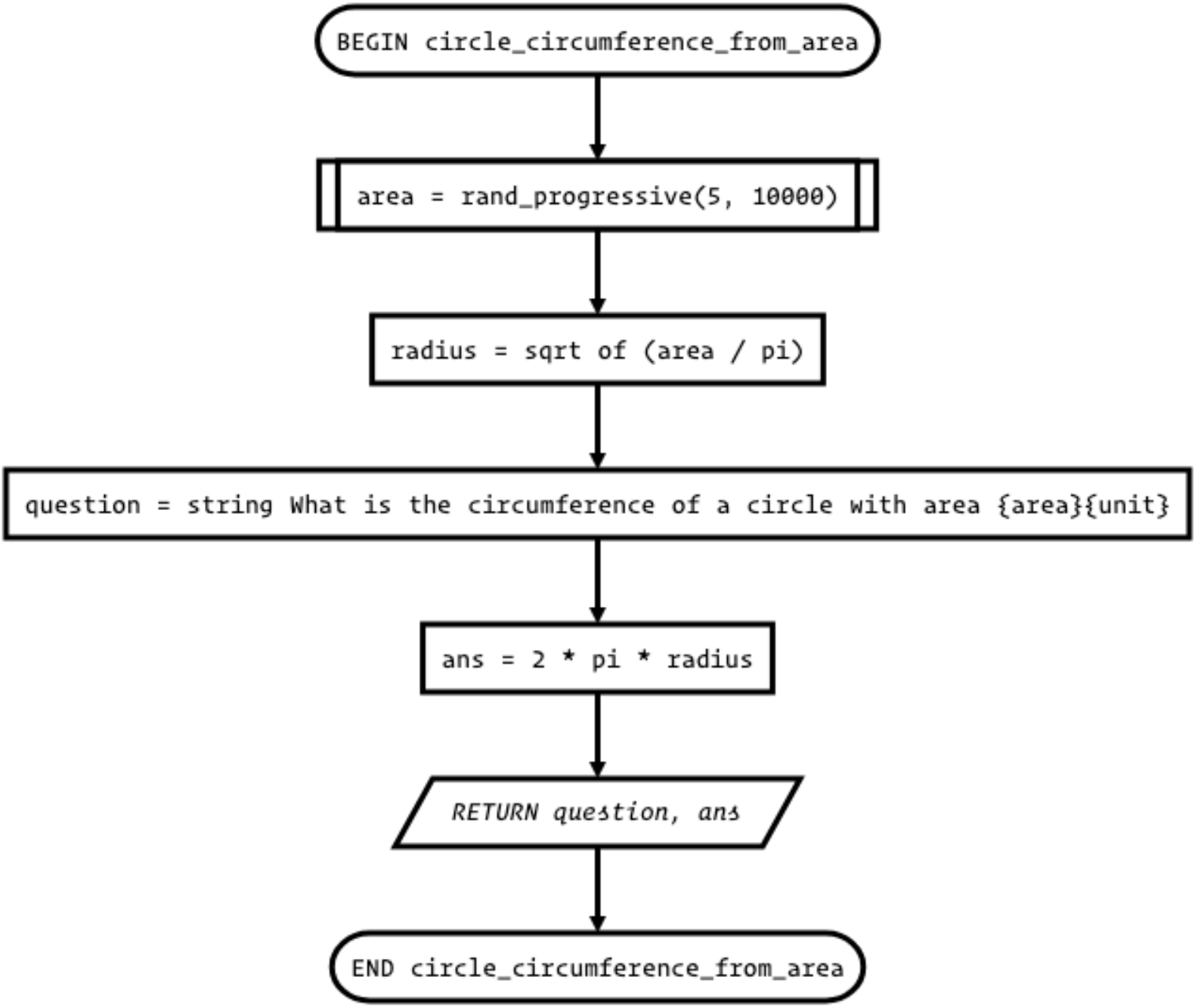

BEGIN circle_area_from_radius
radius = rand_progressive(5, 100)
question = string What is the area of a circle with radius {radius}{unit}
ans = pi * radius ** 2
RETURN question, ans
END circle_area_from_radius

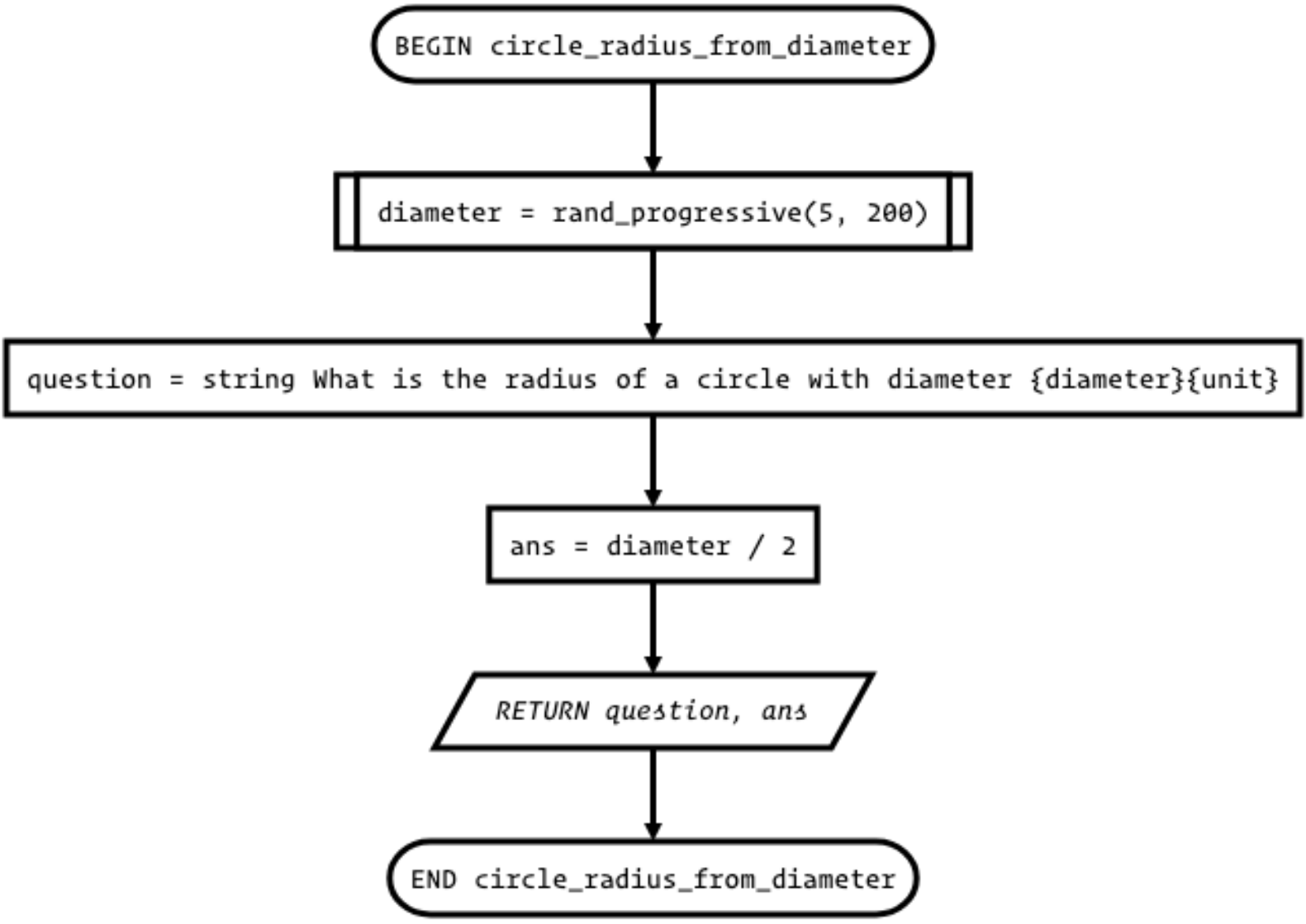

BEGIN circle_area_from_diameter
diameter = rand_progressive(5, 200)
question = string What is the area of a circle with diameter {diameter}{unit}
ans = pi * (diameter / 2) ** 2
RETURN question, ans
END circle_area_from_diameter

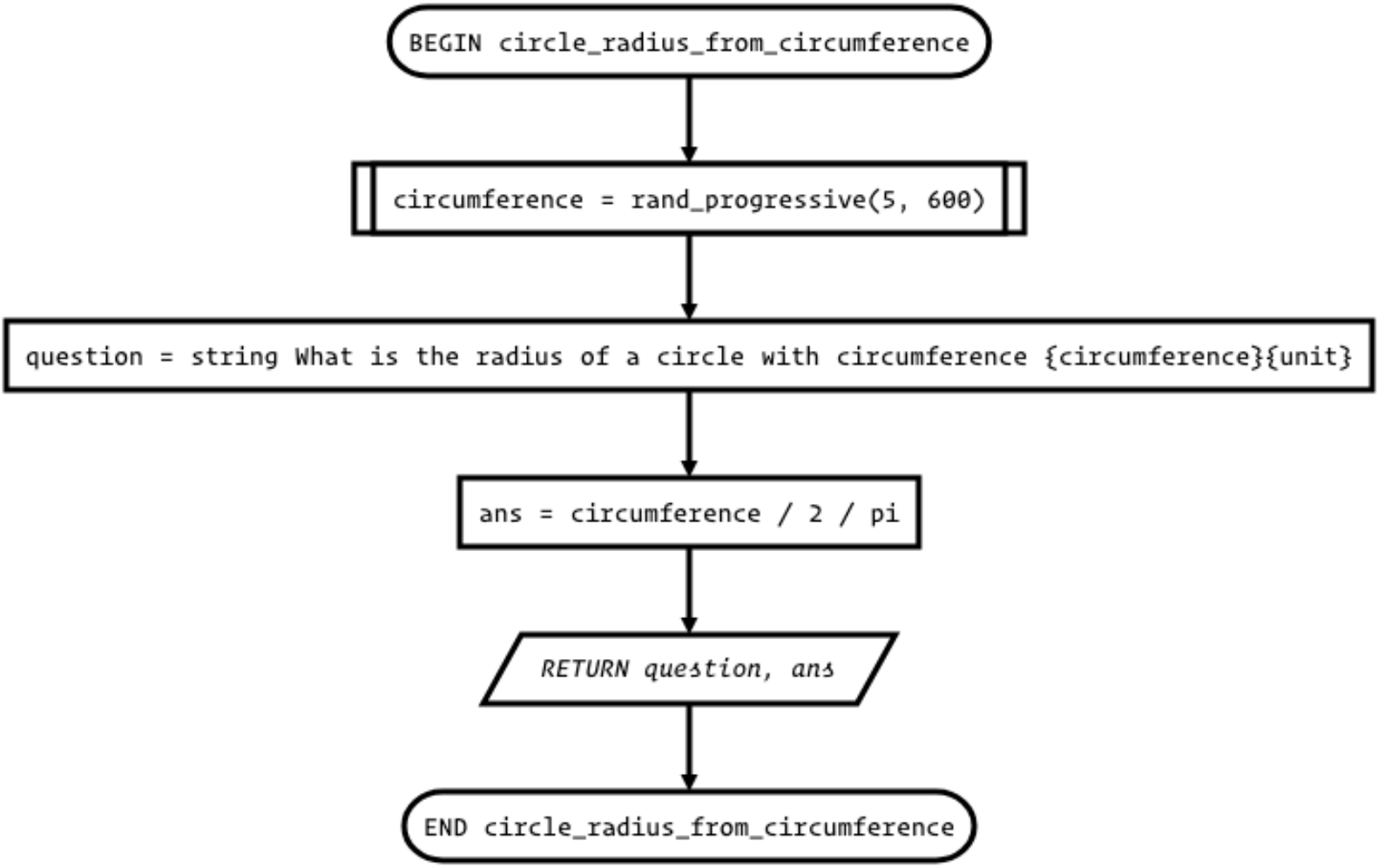

BEGIN circle_area_from_circumference
circumference = rand_progressive(5, 600)
question = string What is the area of a circle with circumference {circumference}{unit}
ans = pi * radius ** 2
RETURN question, ans
END circle_area_from_circumference

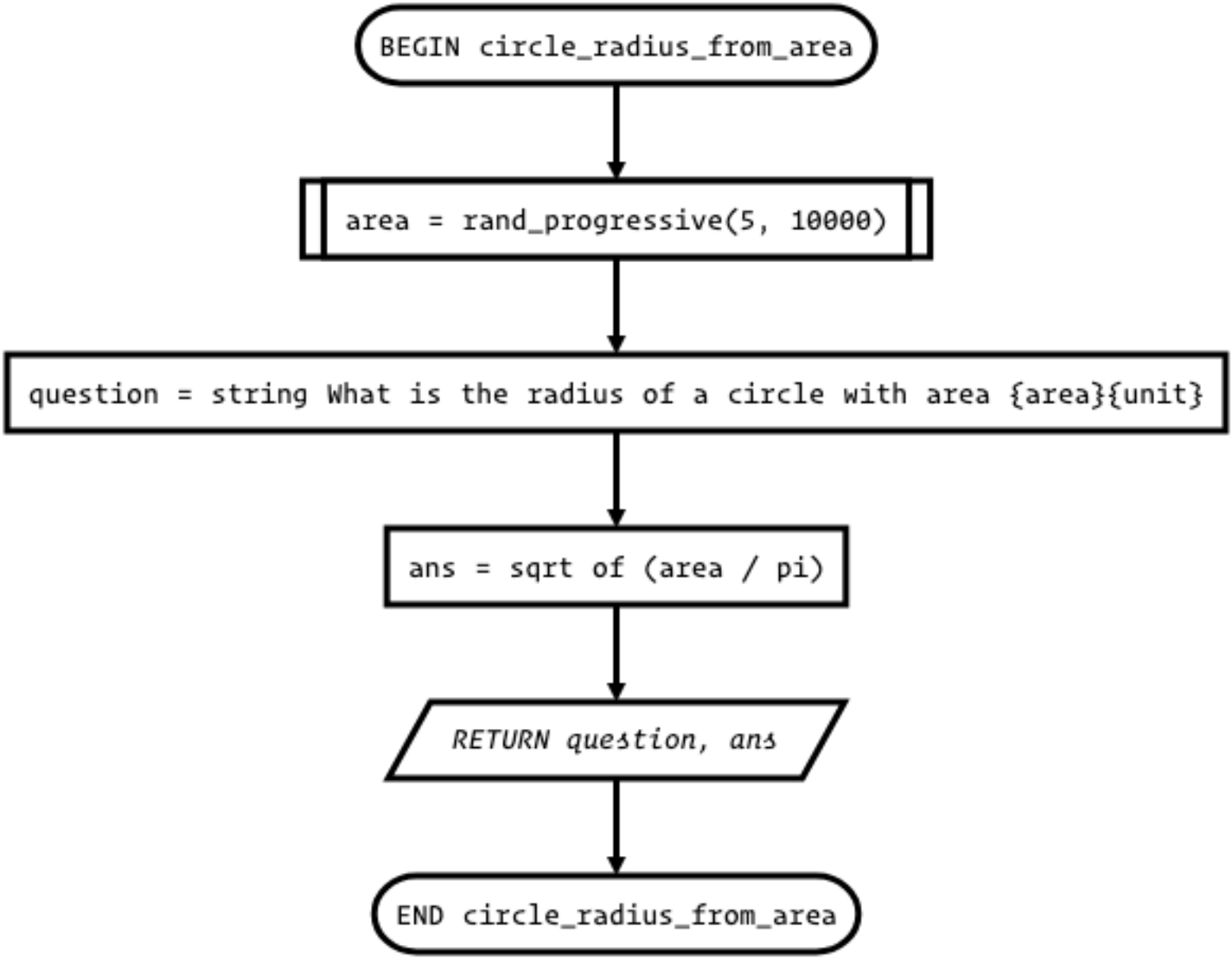

BEGIN circle_circumference_from_radius
radius = rand_progressive(5, 100)
question = string What is the circumference of a circle with radius {radius}{unit}
ans = 2 * pi * radius
RETURN question, ans
END circle_circumference_from_radius

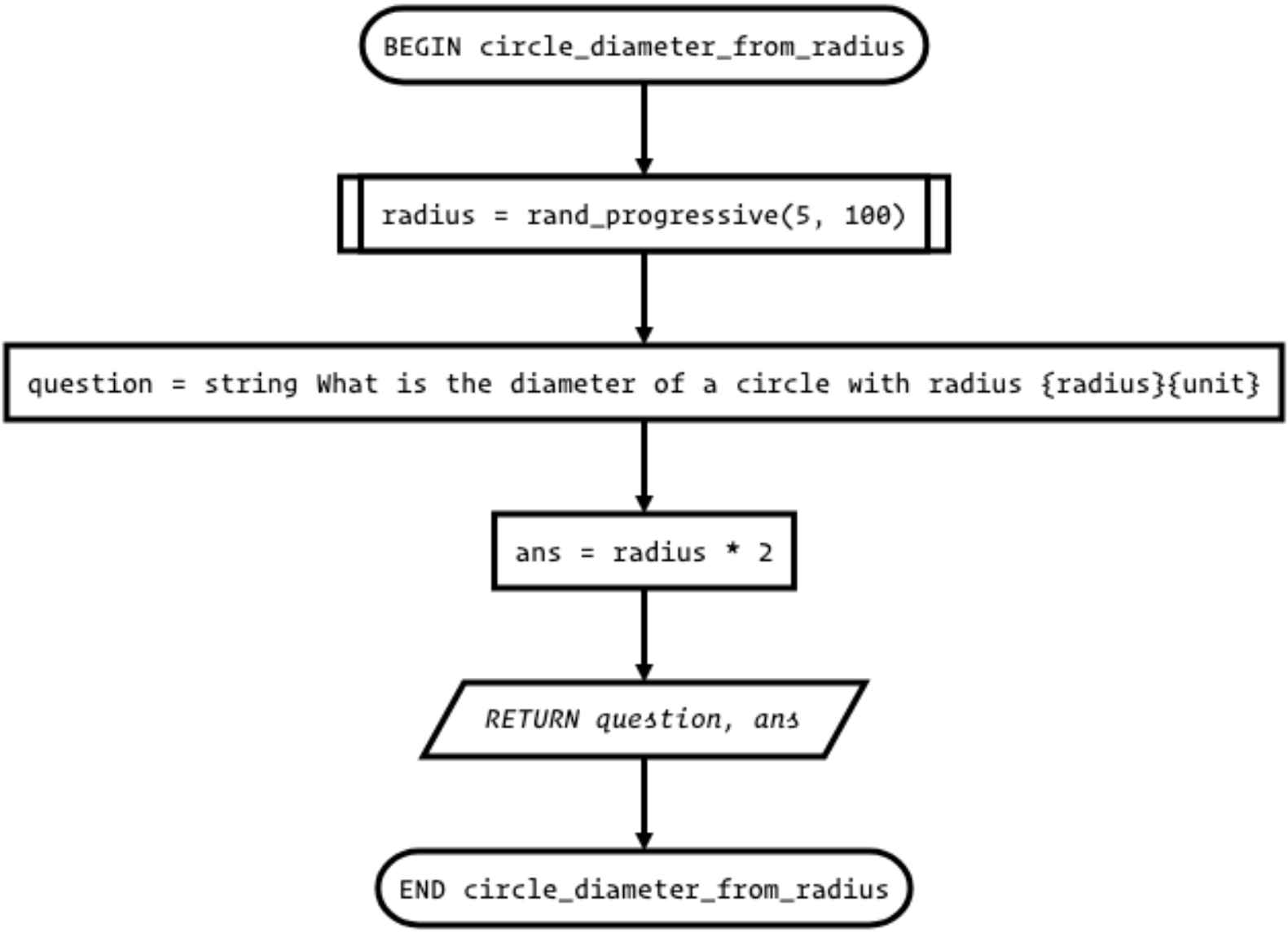

BEGIN circle_circumference_from_diameter
diameter = rand_progressive(5, 200)
question = string What is the circumference of a circle with diameter {diameter}{unit}
ans = 2 * pi * (diameter / 2)
RETURN question, ans
END circle_circumference_from_diameter

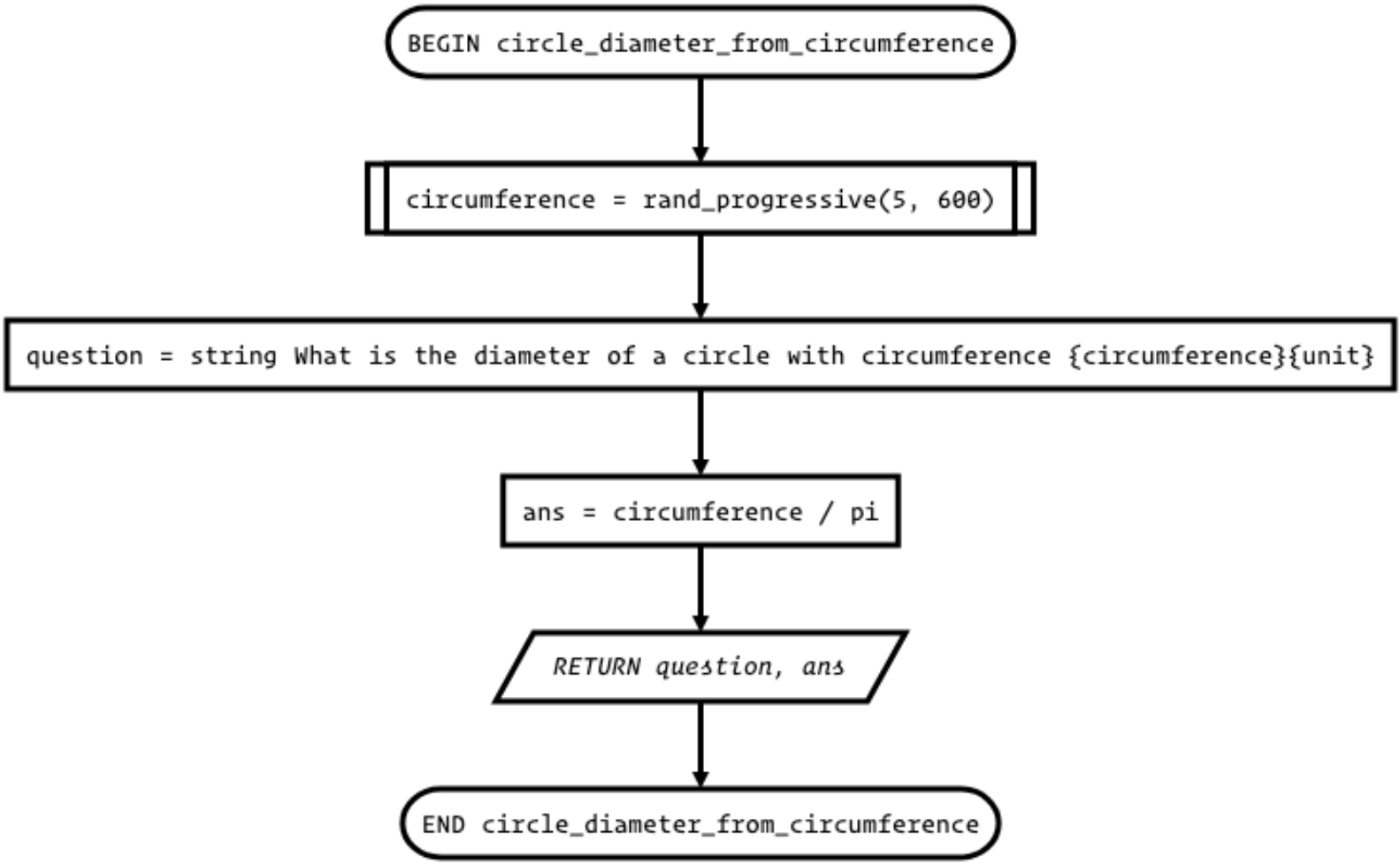

BEGIN circle_circumference_from_area
area = rand_progressive(5, 10000)
radius = sqrt of (area / pi)
question = string What is the circumference of a circle with area {area}{unit}
ans = 2 * pi * radius
RETURN question, ans
END circle_circumference_from_area

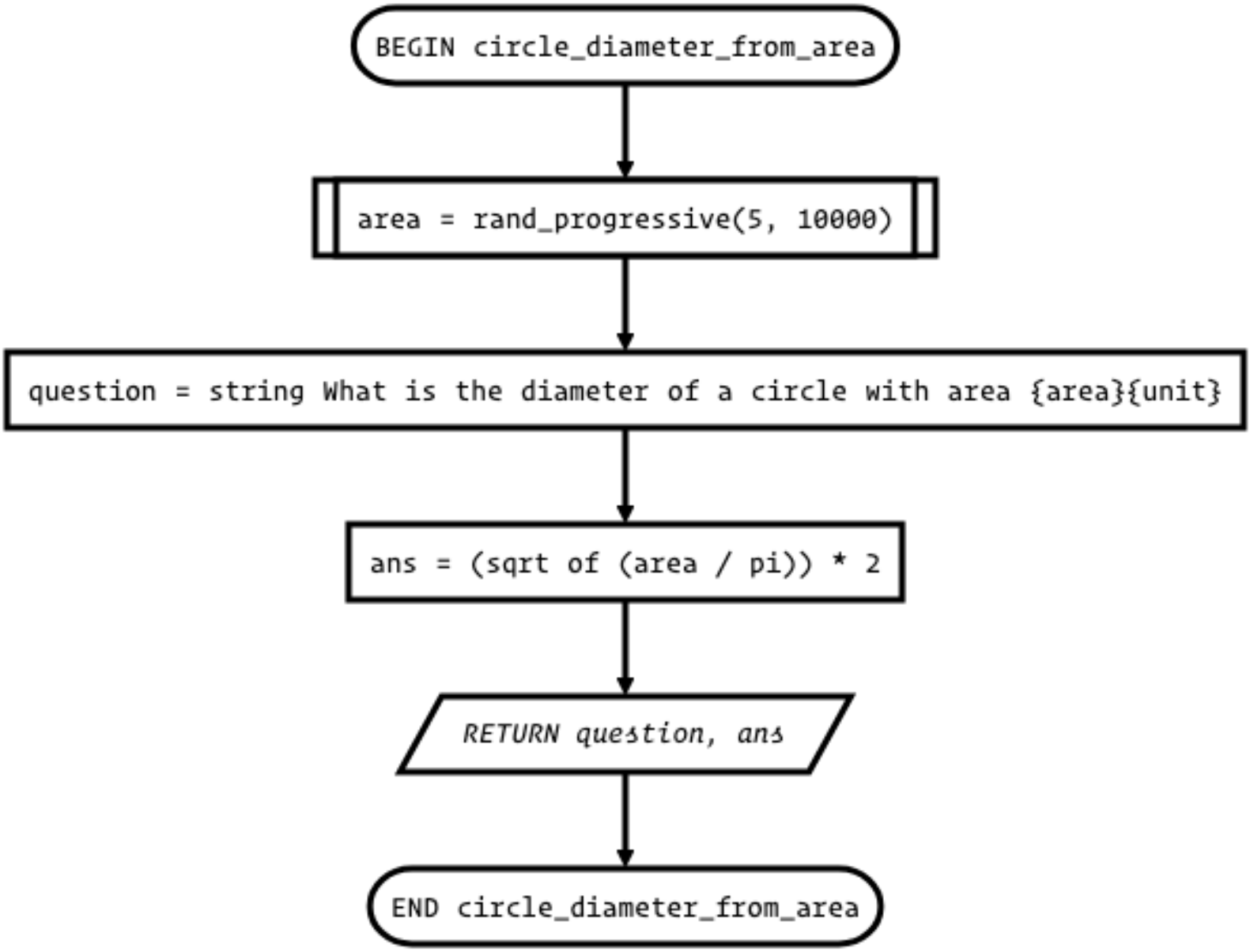

BEGIN circle_radius_from_diameter
diameter = rand_progressive(5, 200)
question = string What is the radius of a circle with diameter {diameter}{unit}
ans = diameter / 2
RETURN question, ans
END circle_radius_from_diameter

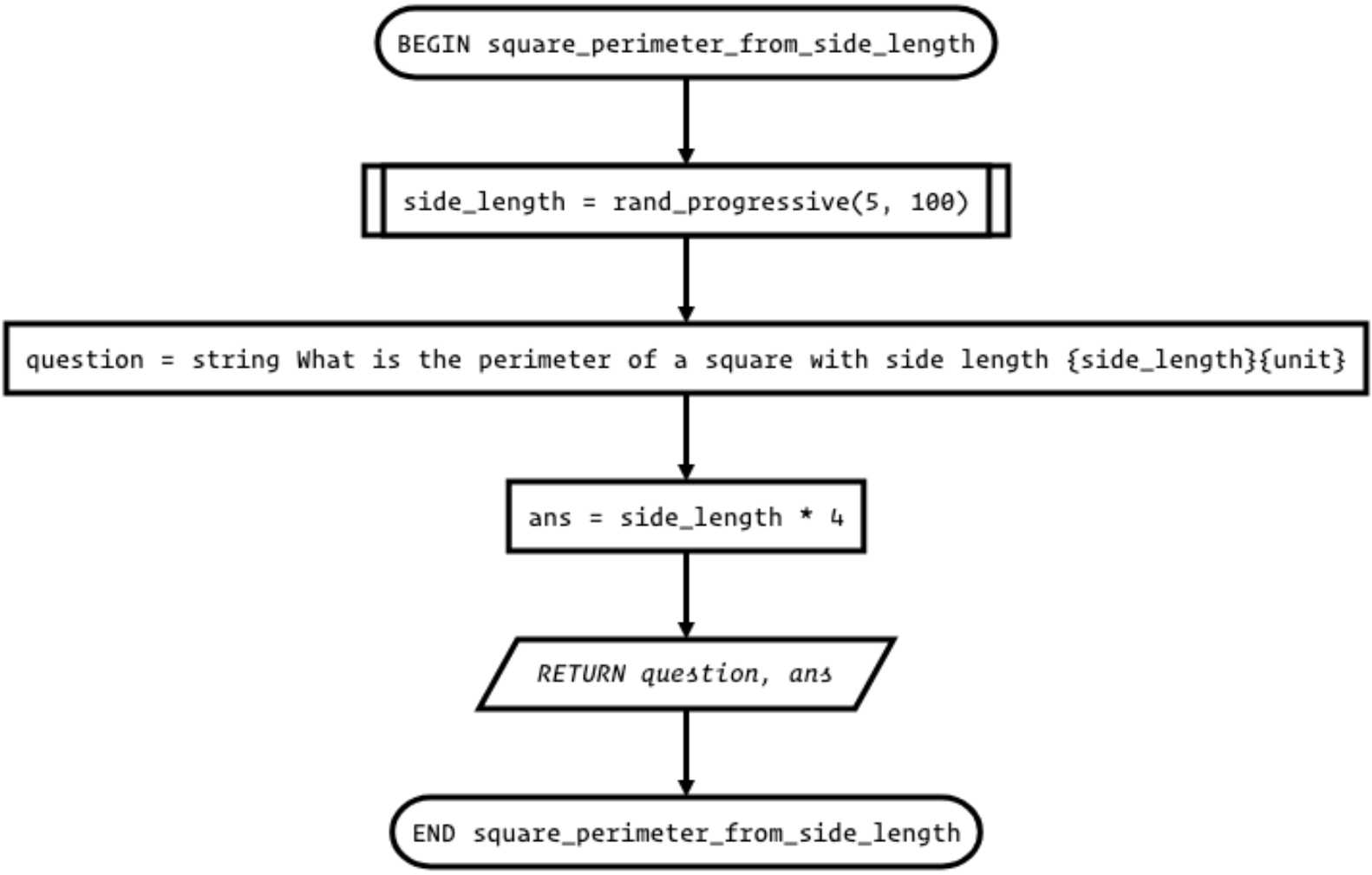

BEGIN circle_radius_from_circumference
circumference = rand_progressive(5, 600)
question = string What is the radius of a circle with circumference {circumference}{unit}
ans = circumference / 2 / pi
RETURN question, ans
END circle_radius_from_circumference

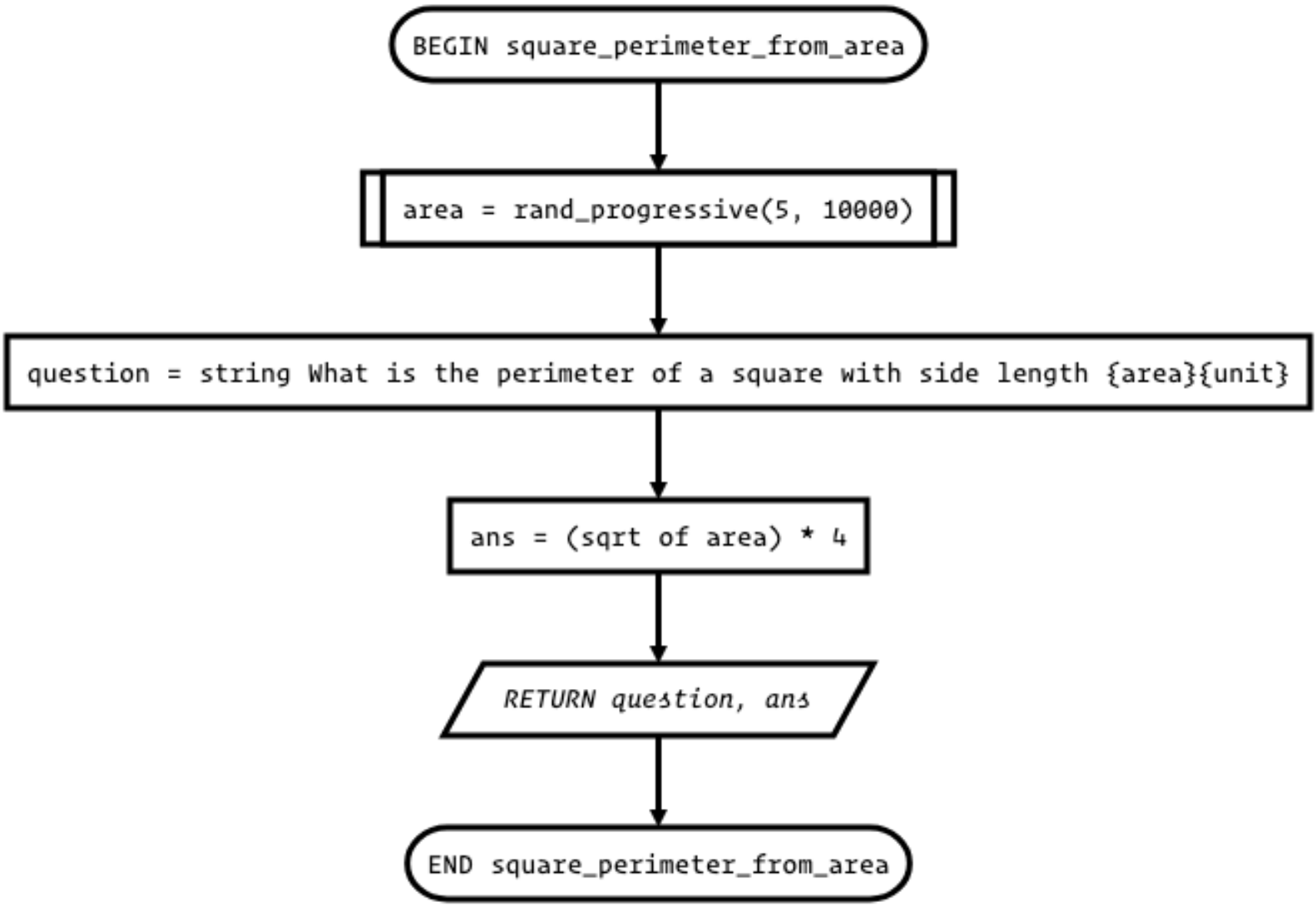

BEGIN circle_radius_from_area
area = rand_progressive(5, 10000)
question = string What is the radius of a circle with area {area}{unit}
ans = sqrt of (area / pi)
RETURN question, ans
END circle_radius_from_area

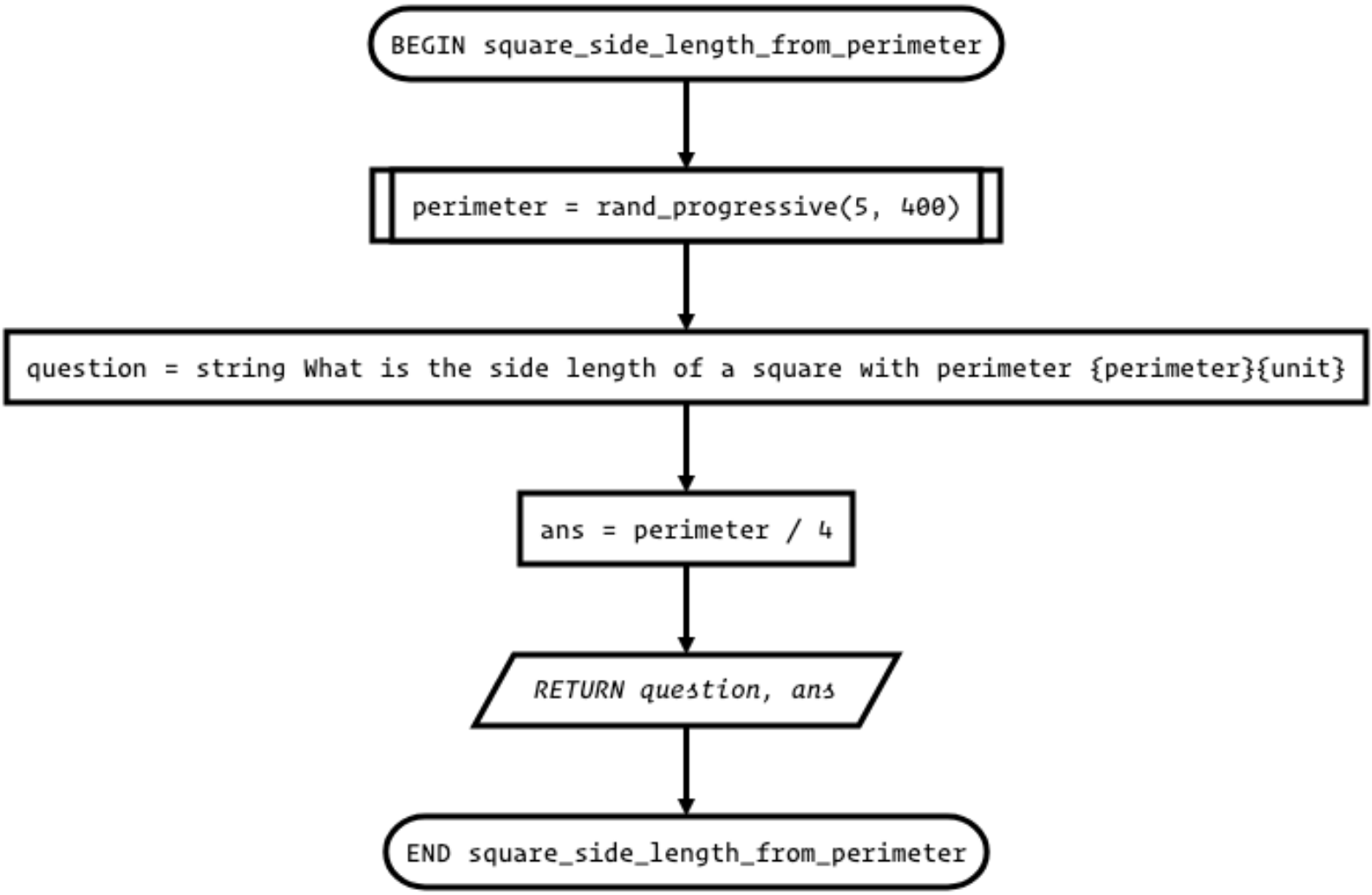

BEGIN circle_diameter_from_radius
radius = rand_progressive(5, 100)
question = string What is the diameter of a circle with radius {radius}{unit}
ans = radius * 2
RETURN question, ans
END circle_diameter_from_radius

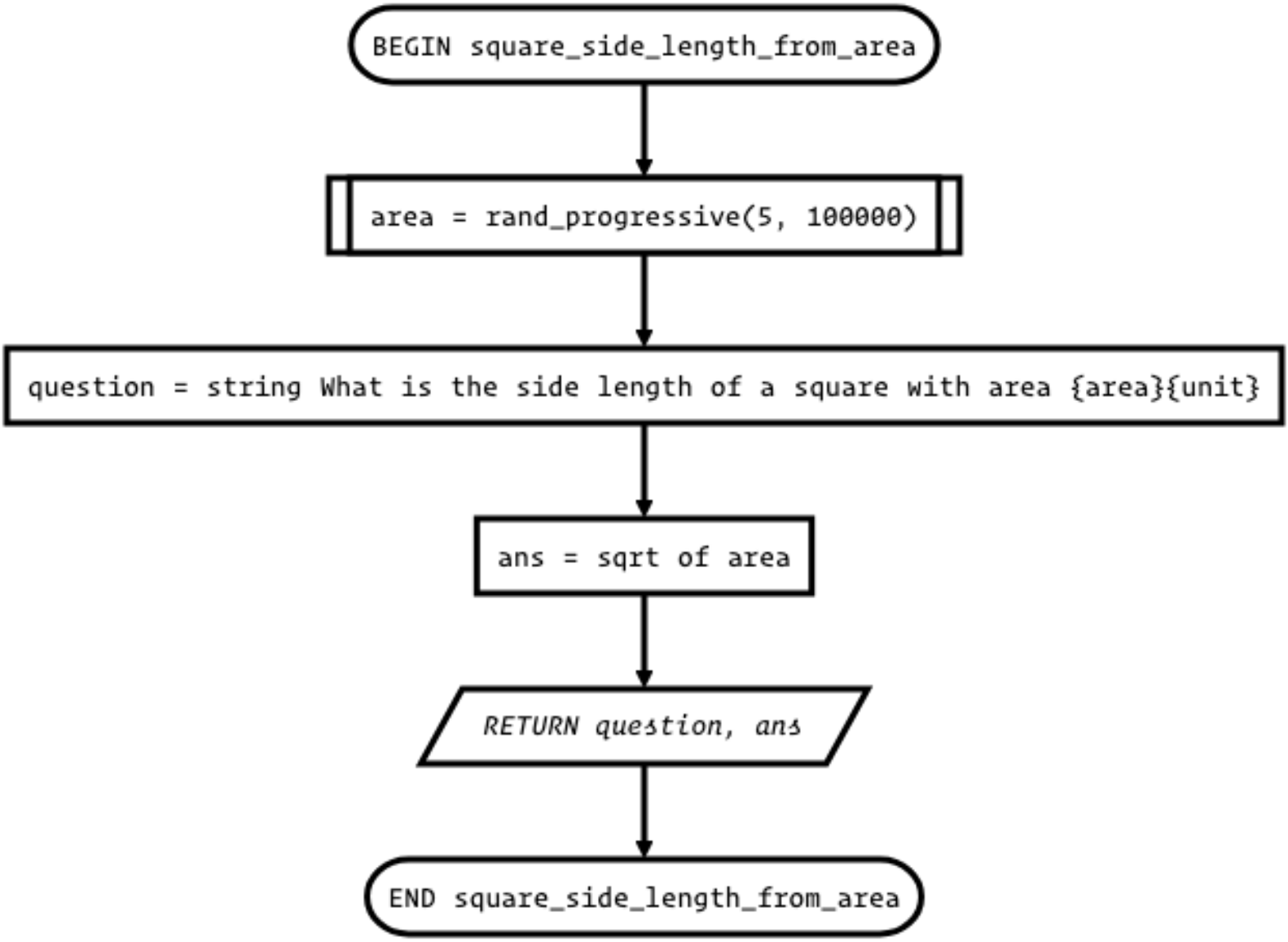

BEGIN circle_diameter_from_circumference
circumference = rand_progressive(5, 600)
question = string What is the diameter of a circle with circumference {circumference}{unit}
ans = circumference / pi
RETURN question, ans
END circle_diameter_from_circumference

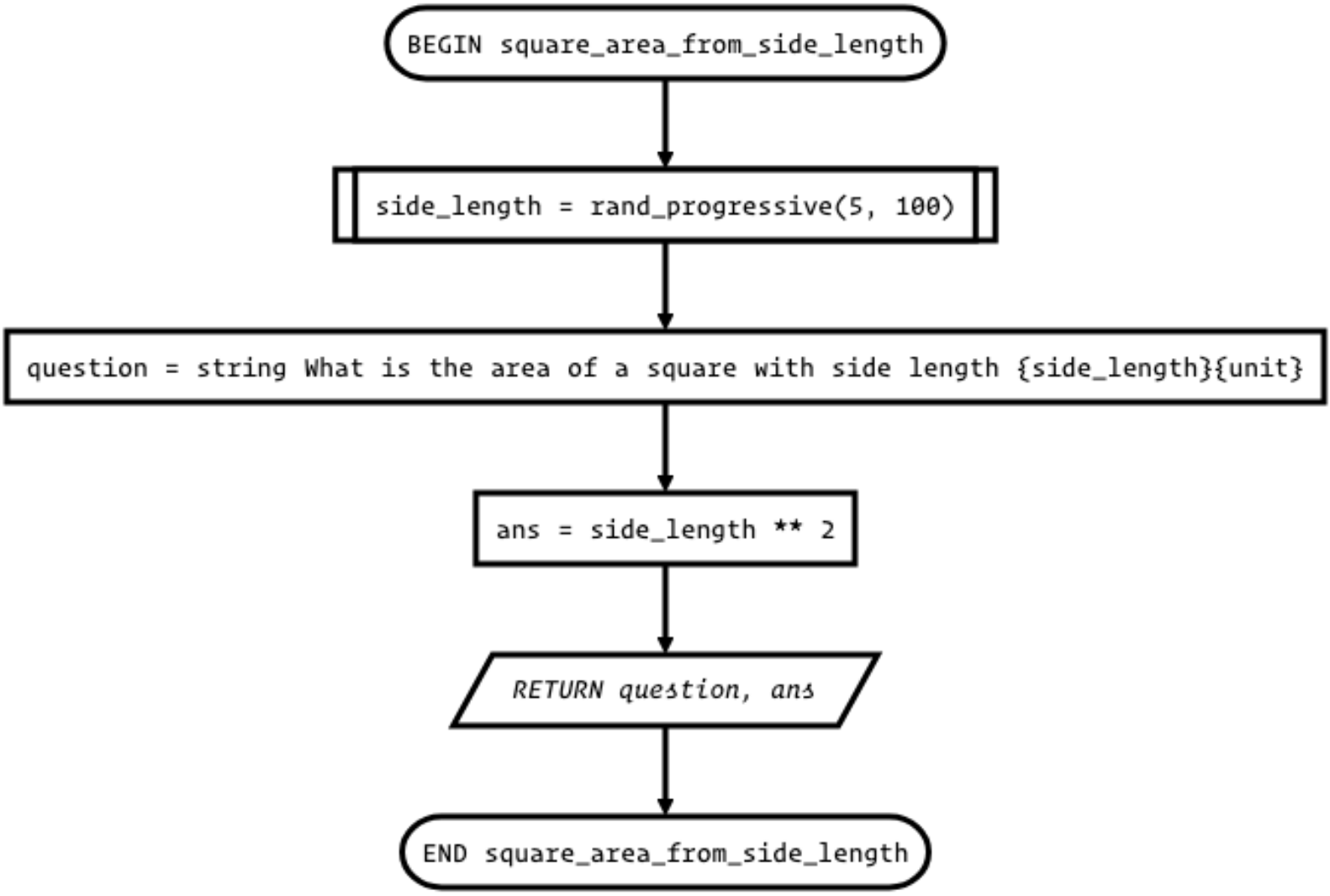

BEGIN circle_diameter_from_area
area = rand_progressive(5, 10000)
question = string What is the diameter of a circle with area {area}{unit}
ans = (sqrt of (area / pi)) * 2
RETURN question, ans
END circle_diameter_from_area

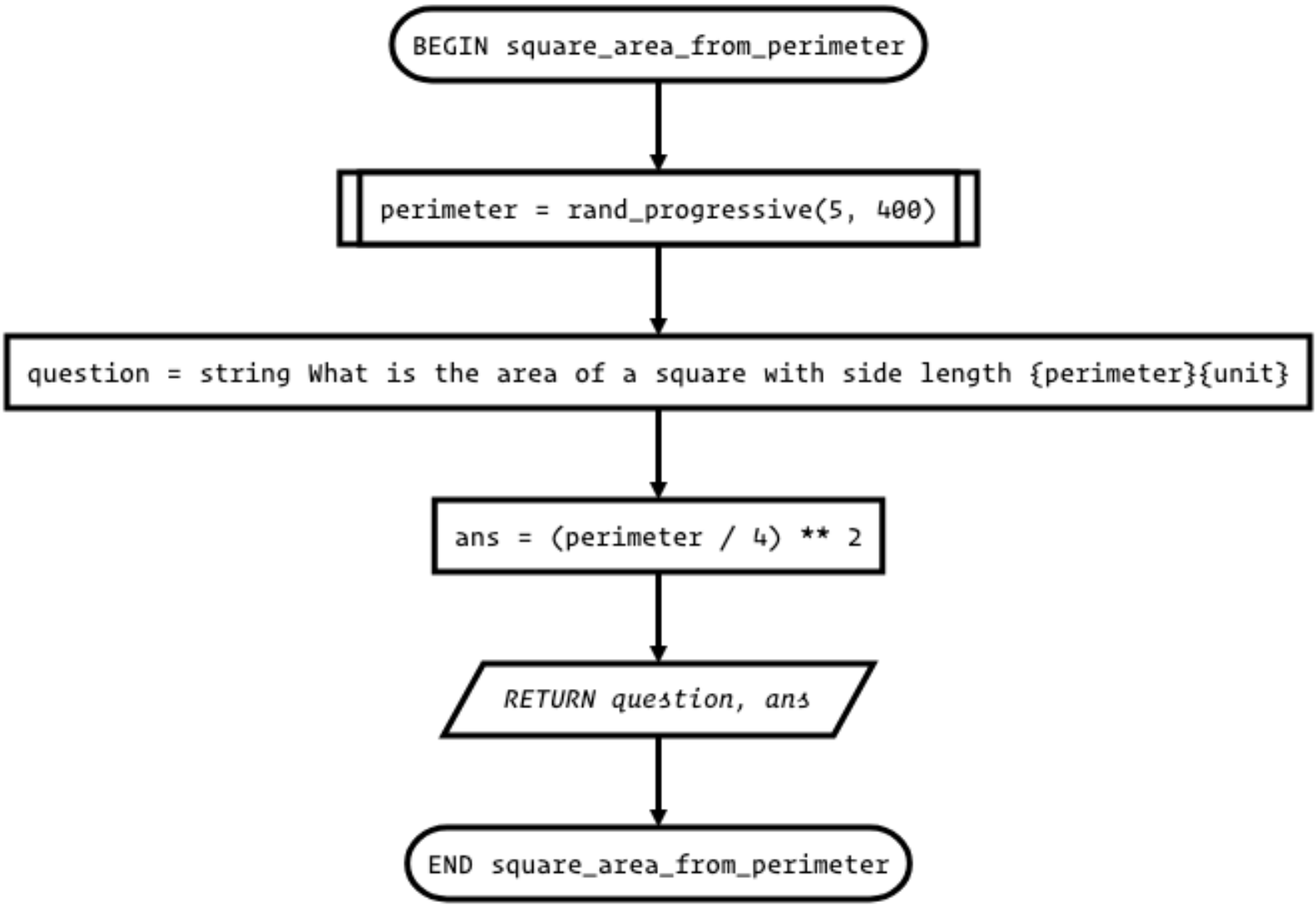

BEGIN square_perimeter_from_side_length
side_length = rand_progressive(5, 100)
question = string What is the perimeter of a square with side length {side_length}{unit}
ans = side_length * 4
RETURN question, ans
END square_perimeter_from_side_length

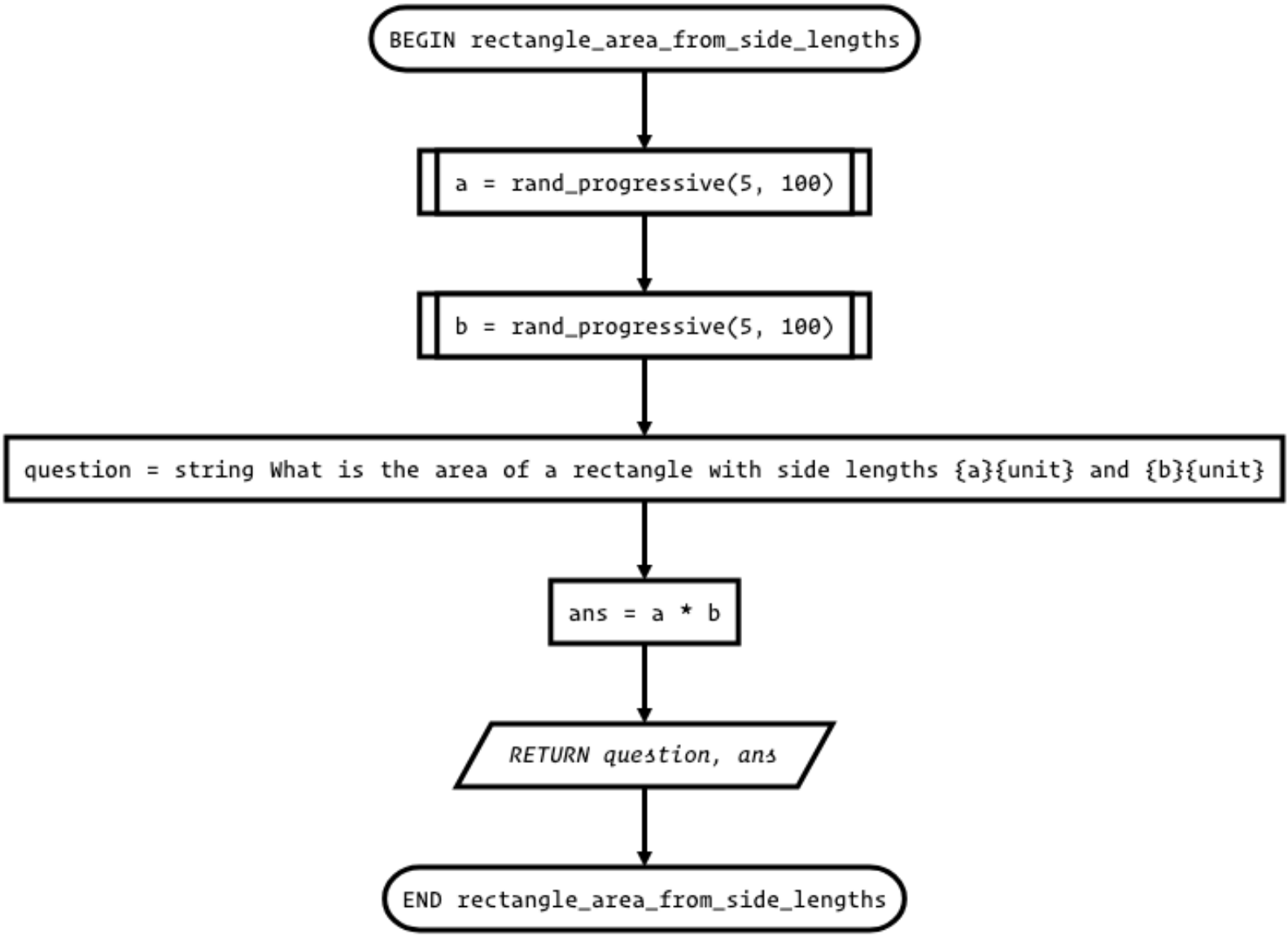

BEGIN square_perimeter_from_area
area = rand_progressive(5, 10000)
question = string What is the perimeter of a square with side length {area}{unit}
ans = (sqrt of area) * 4
RETURN question, ans
END square_perimeter_from_area

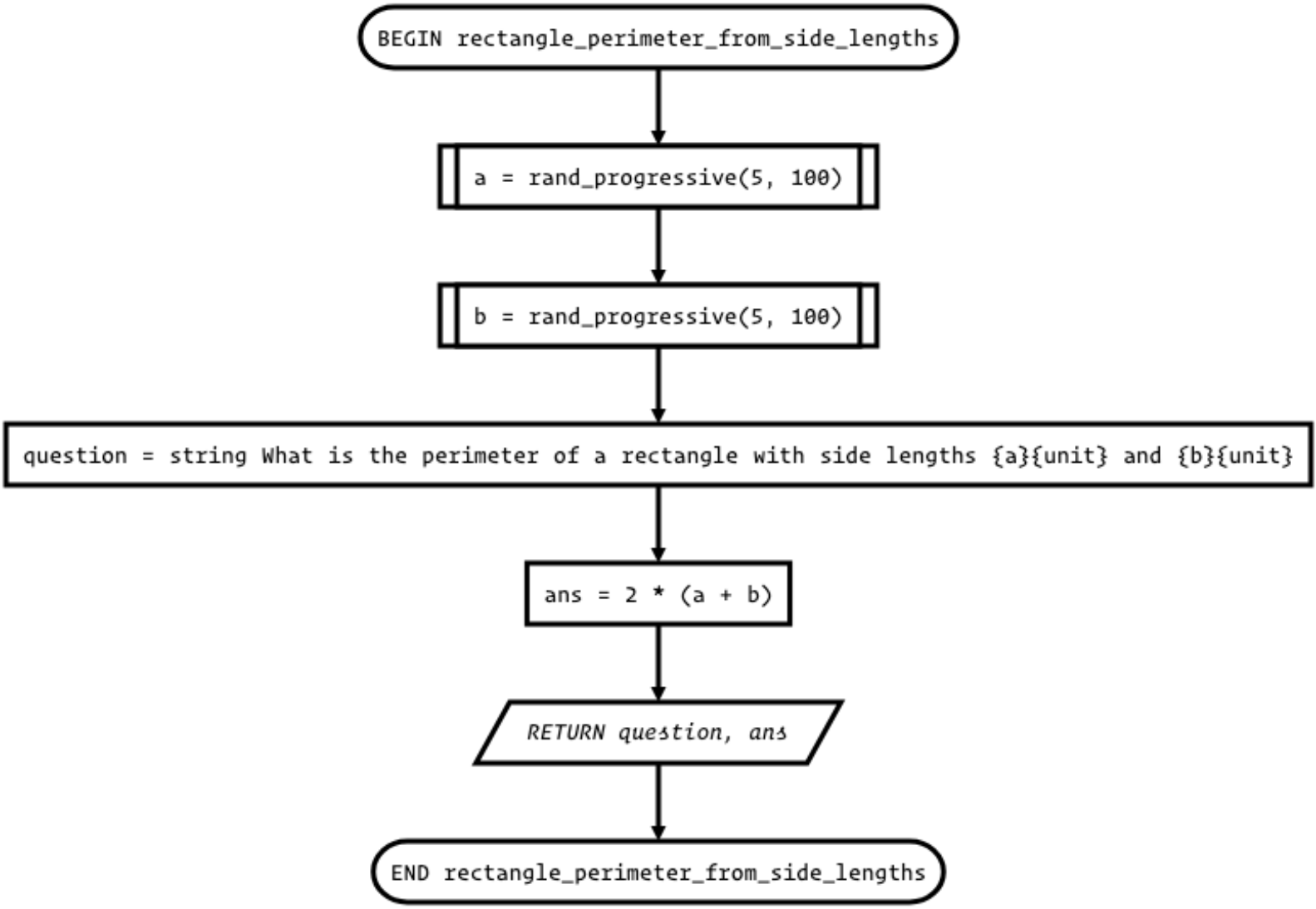

BEGIN square_side_length_from_perimeter
perimeter = rand_progressive(5, 400)
question = string What is the side length of a square with perimeter {perimeter}{unit}
ans = perimeter / 4
RETURN question, ans
END square_side_length_from_perimeter

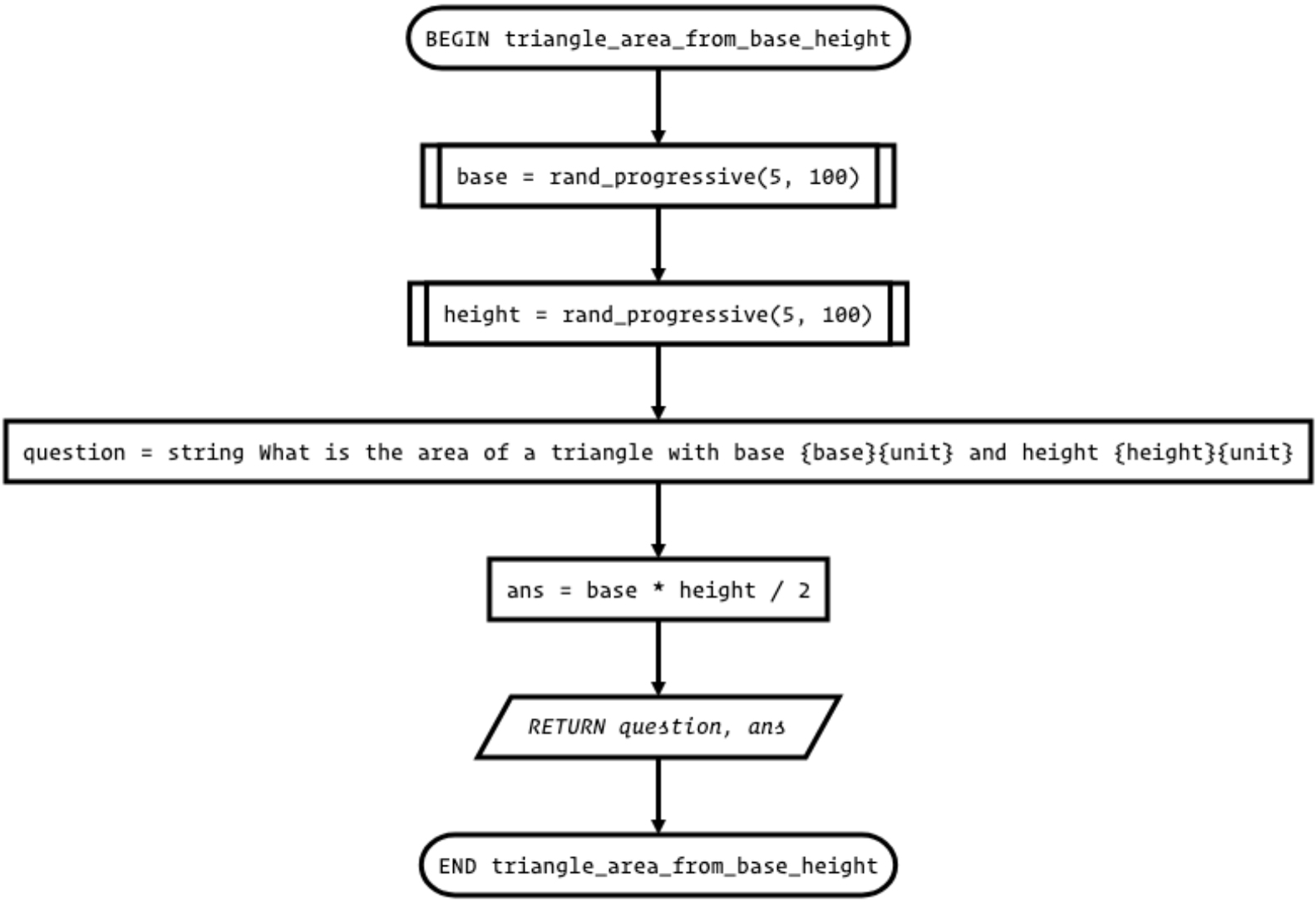

BEGIN square_side_length_from_area
area = rand_progressive(5, 100000)
question = string What is the side length of a square with area {area}{unit}
ans = sqrt of area
RETURN question, ans
END square_side_length_from_area

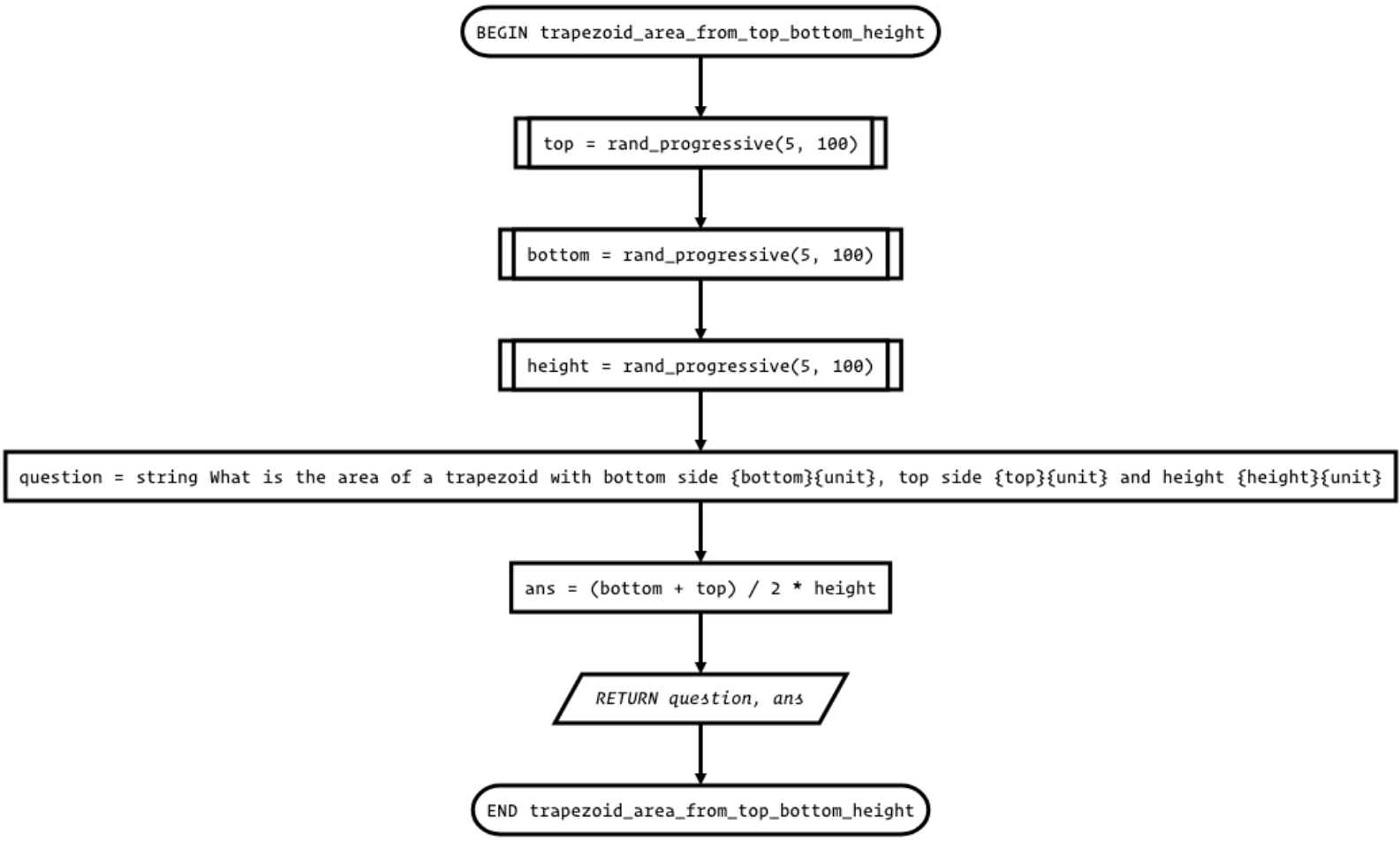

BEGIN square_area_from_side_length
side_length = rand_progressive(5, 100)
question = string What is the area of a square with side length {side_length}{unit}
ans = side_length ** 2
RETURN question, ans
END square_area_from_side_length

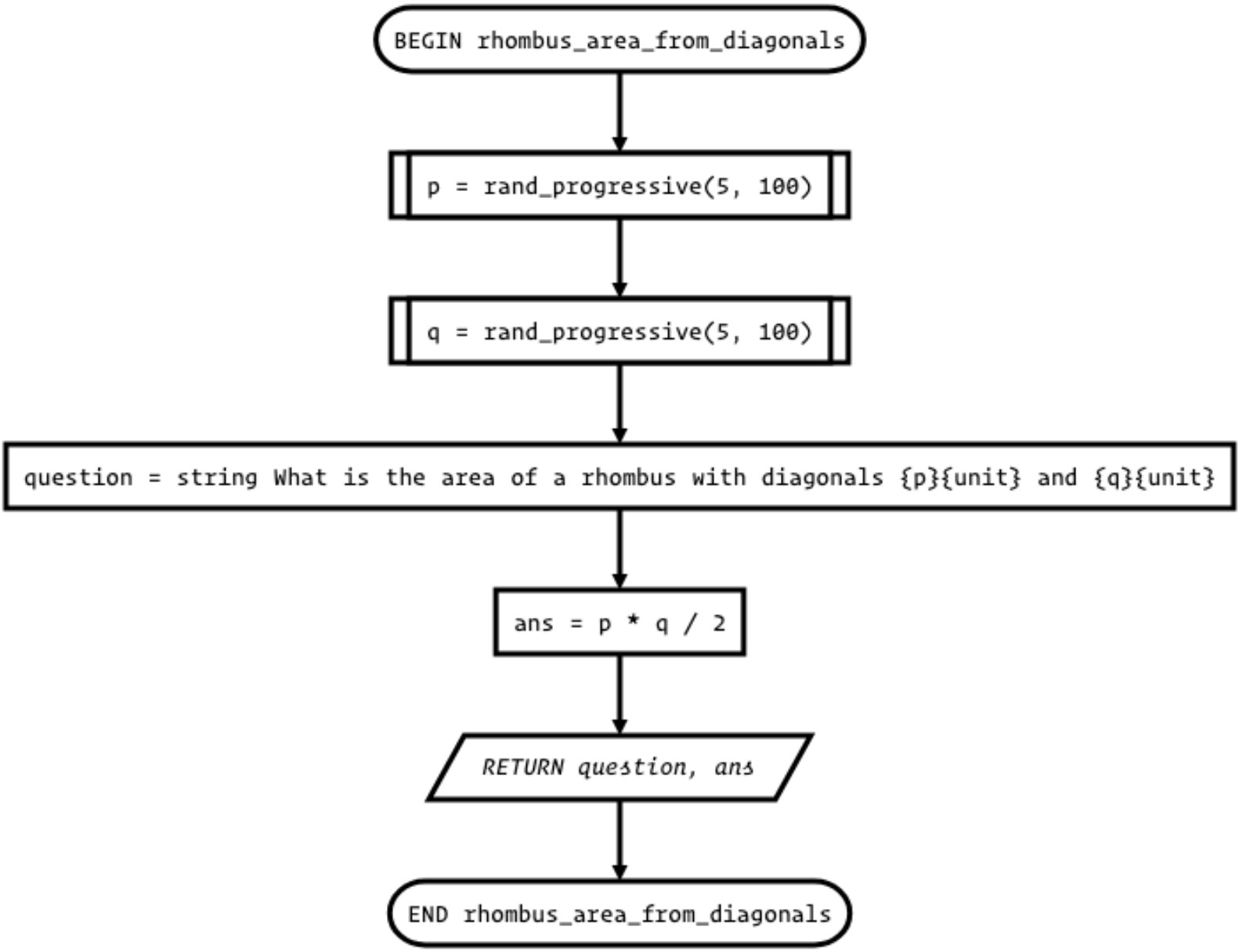

BEGIN square_area_from_perimeter
perimeter = rand_progressive(5, 400)
question = string What is the area of a square with side length {perimeter}{unit}
ans = (perimeter / 4) ** 2
RETURN question, ans
END square_area_from_perimeter

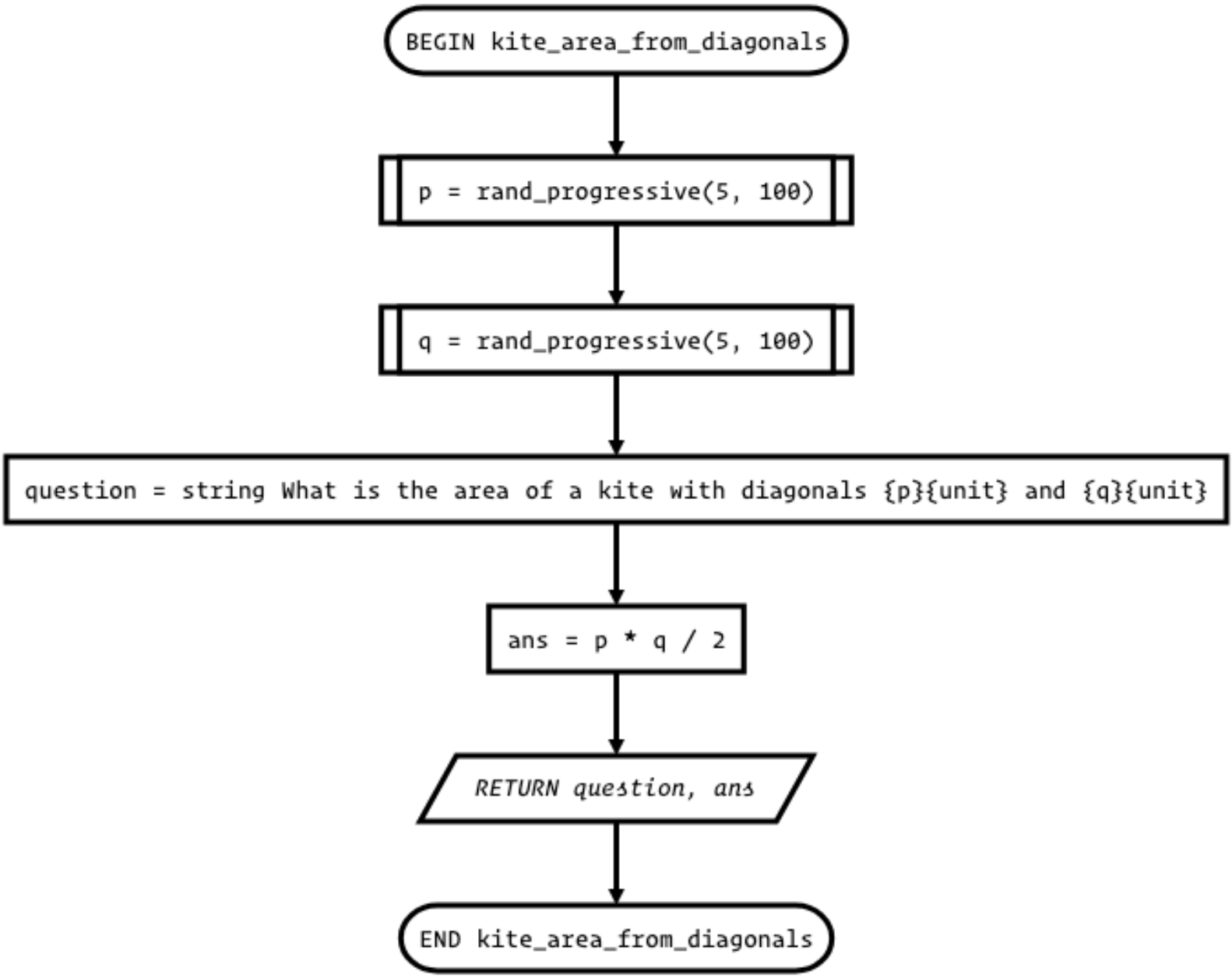

BEGIN rectangle_area_from_side_lengths
a = rand_progressive(5, 100)
b = rand_progressive(5, 100)
question = string What is the area of a rectangle with side lengths {a}{unit} and {b}{unit}
ans = a * b
RETURN question, ans
END rectangle_area_from_side_lengths

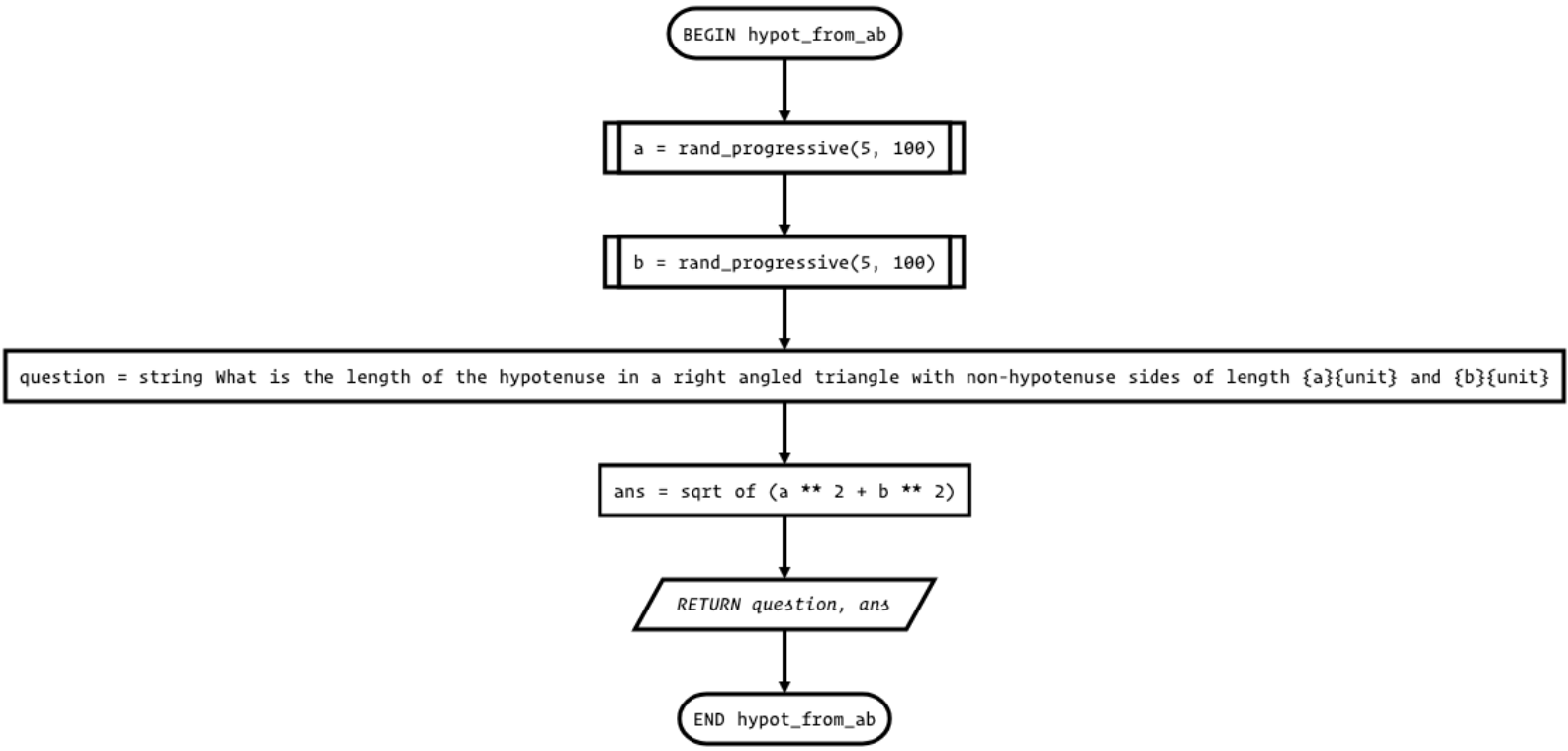

BEGIN rectangle_perimeter_from_side_lengths
a = rand_progressive(5, 100)
b = rand_progressive(5, 100)
question = string What is the perimeter of a rectangle with side lengths {a}{unit} and {b}{unit}
ans = 2 * (a + b)
RETURN question, ans
END rectangle_perimeter_from_side_lengths

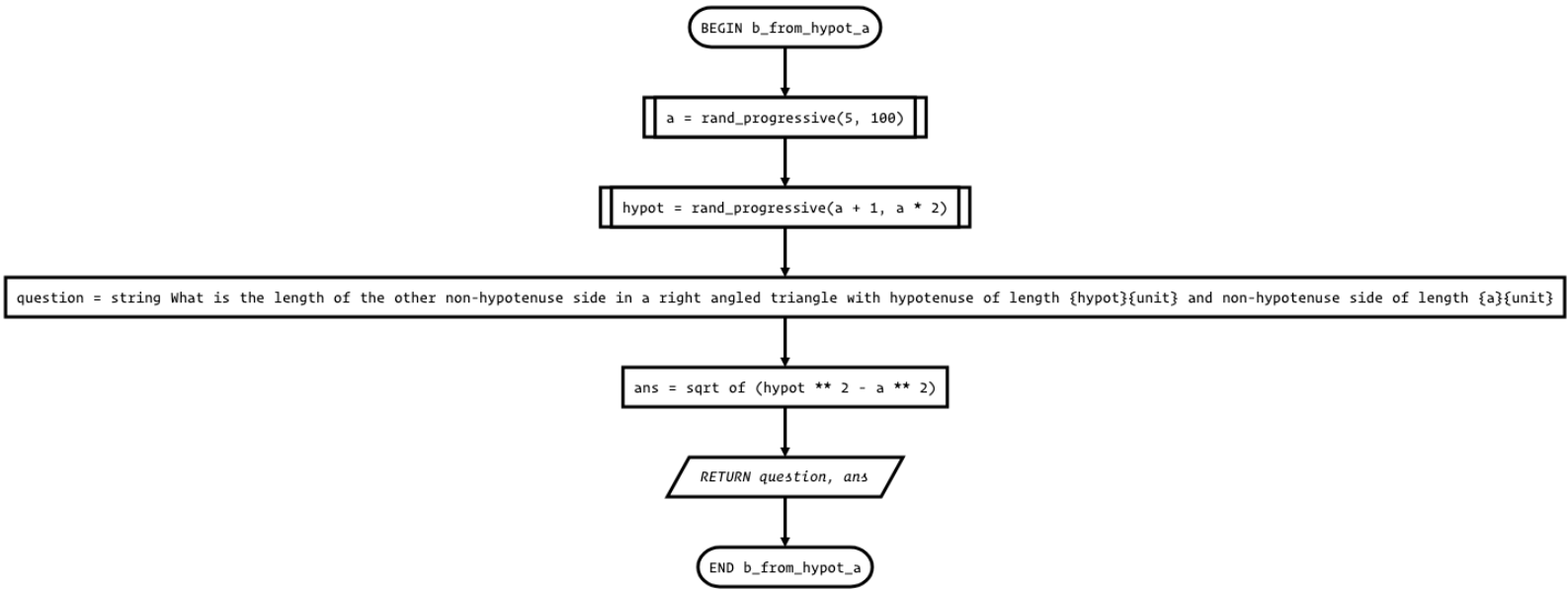

BEGIN triangle_area_from_base_height
base = rand_progressive(5, 100)
height = rand_progressive(5, 100)
question = string What is the area of a triangle with base {base}{unit} and height {height}{unit}
ans = base * height / 2
RETURN question, ans
END triangle_area_from_base_height

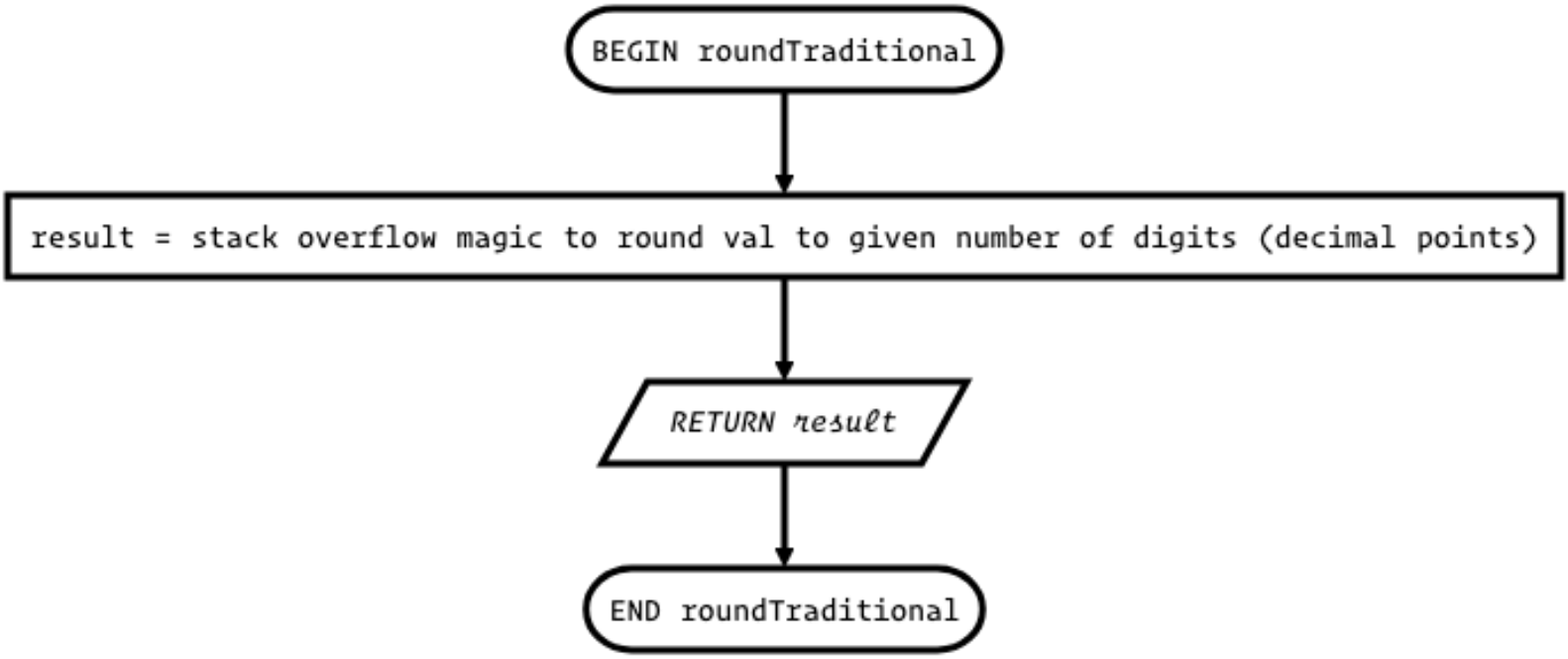

BEGIN trapezoid_area_from_top_bottom_height
top = rand_progressive(5, 100)
bottom = rand_progressive(5, 100)
height = rand_progressive(5, 100)
question = string What is the area of a trapezoid with bottom side {bottom}{unit}, top side {top}{unit} and height {height}{unit}
ans = (bottom + top) / 2 * height
RETURN question, ans
END trapezoid_area_from_top_bottom_height

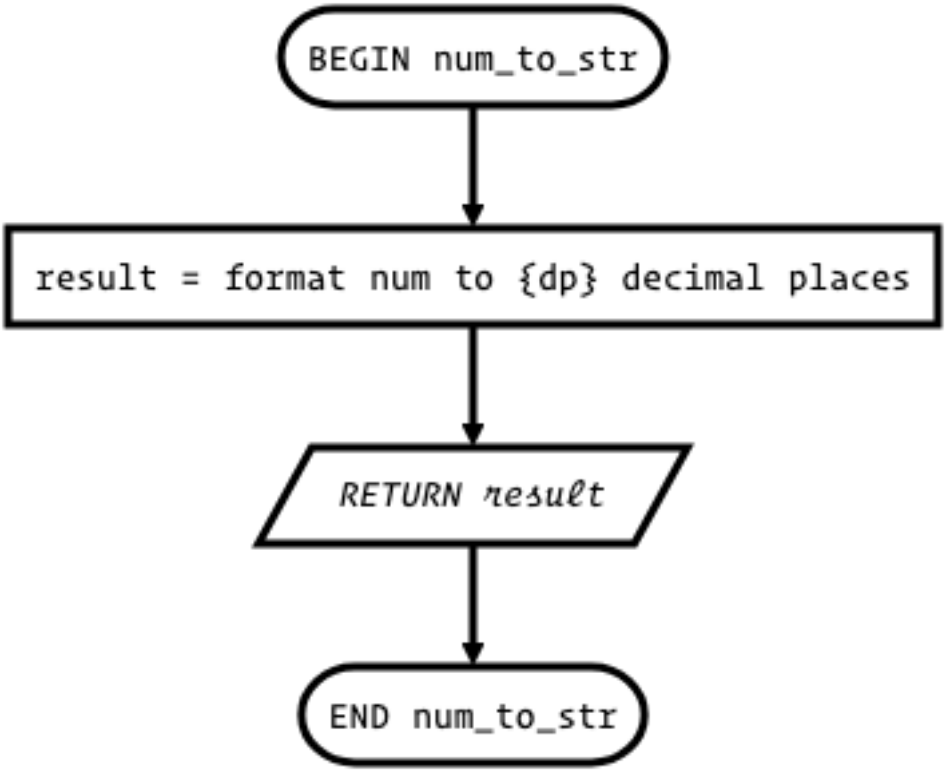

BEGIN rhombus_area_from_diagonals
p = rand_progressive(5, 100)
q = rand_progressive(5, 100)
question = string What is the area of a rhombus with diagonals {p}{unit} and {q}{unit}
ans = p * q / 2
RETURN question, ans
END rhombus_area_from_diagonals

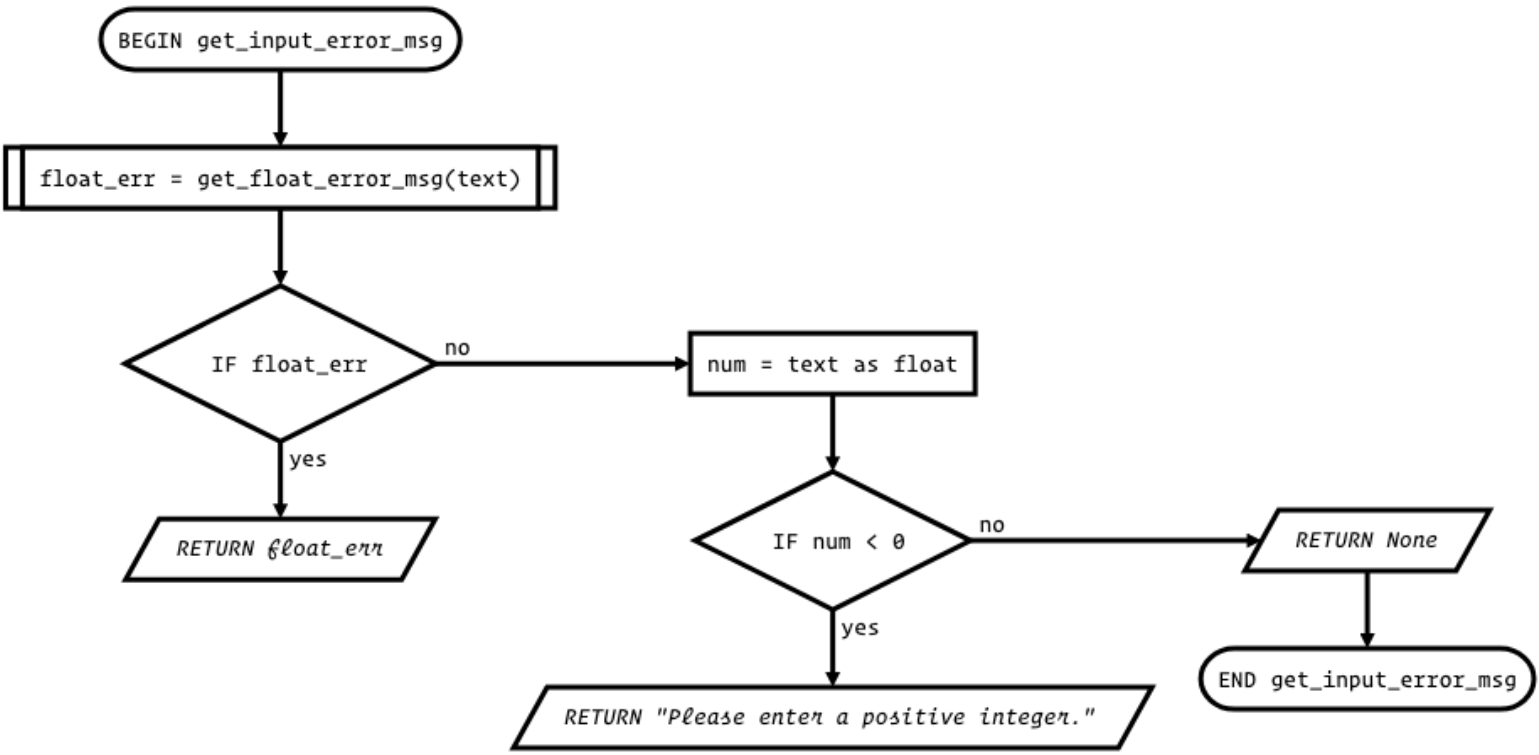

BEGIN kite_area_from_diagonals
p = rand_progressive(5, 100)
q = rand_progressive(5, 100)
question = string What is the area of a kite with diagonals {p}{unit} and {q}{unit}
ans = p * q / 2
RETURN question, ans
END kite_area_from_diagonals

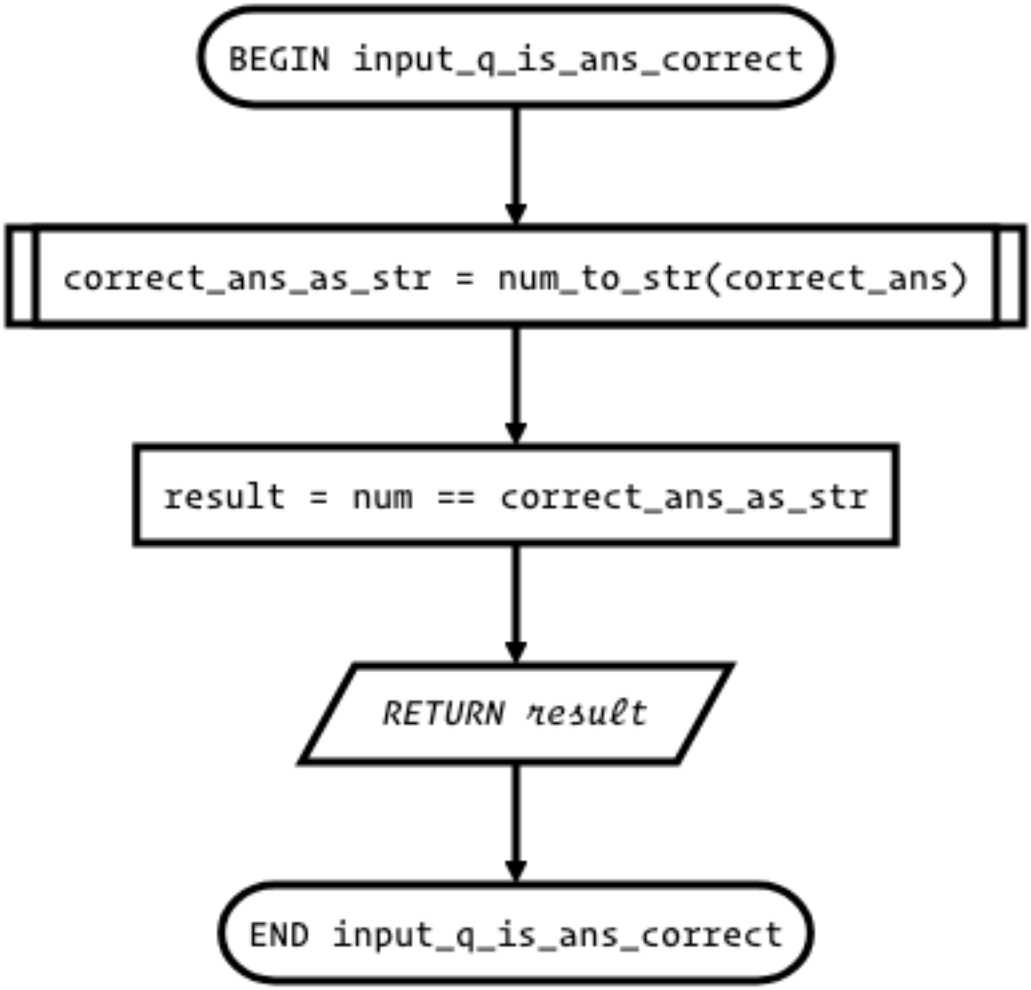

BEGIN hypot_from_ab
a = rand_progressive(5, 100)
b = rand_progressive(5, 100)
question = string What is the length of the hypotenuse in a right angled triangle with non-hypotenuse sides of length {a}{unit} and {b}{unit}
ans = sqrt of (a ** 2 + b ** 2)
RETURN question, ans
END hypot_from_ab

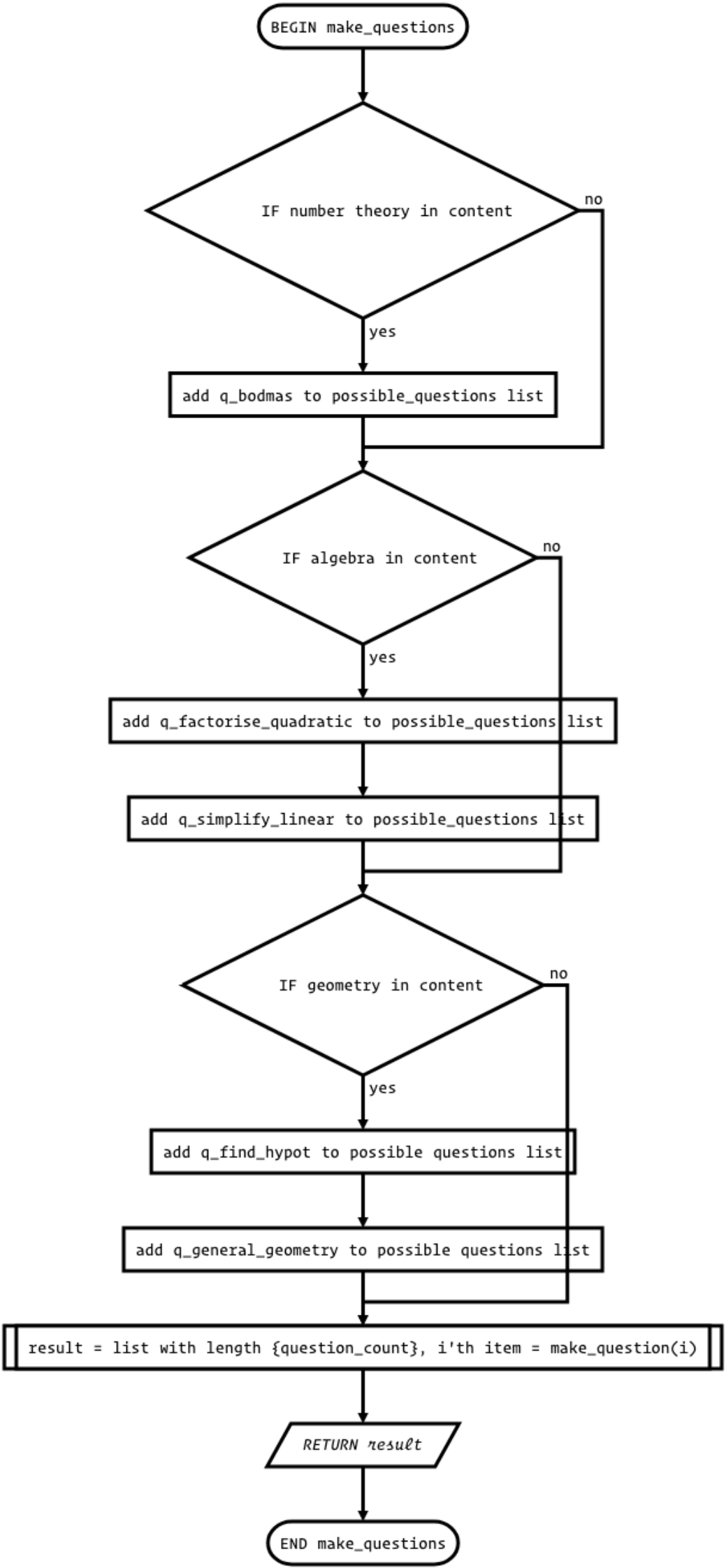

BEGIN b_from_hypot_a
a = rand_progressive(5, 100)
hypot = rand_progressive(a + 1, a * 2)
question = string What is the length of the other non-hypotenuse side in a right angled triangle with hypotenuse of length {hypot}{unit} and non-hypotenuse side of length {a}{unit}
ans = sqrt of (hypot ** 2 - a ** 2)
RETURN question, ans
END b_from_hypot_a

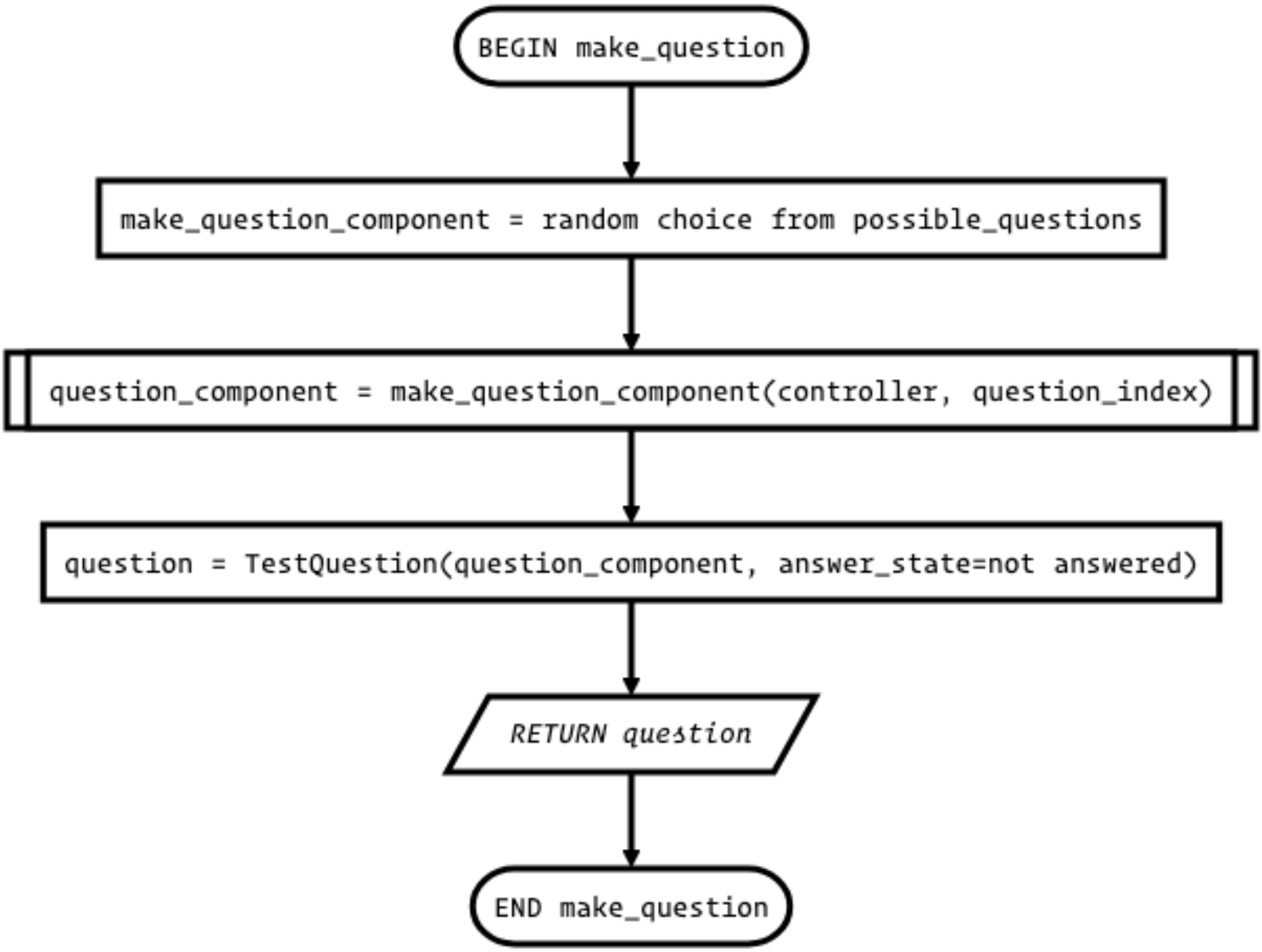

BEGIN roundTraditional
result = stack overflow magic to round val to given number of digits (decimal points)
RETURN result
END roundTraditional

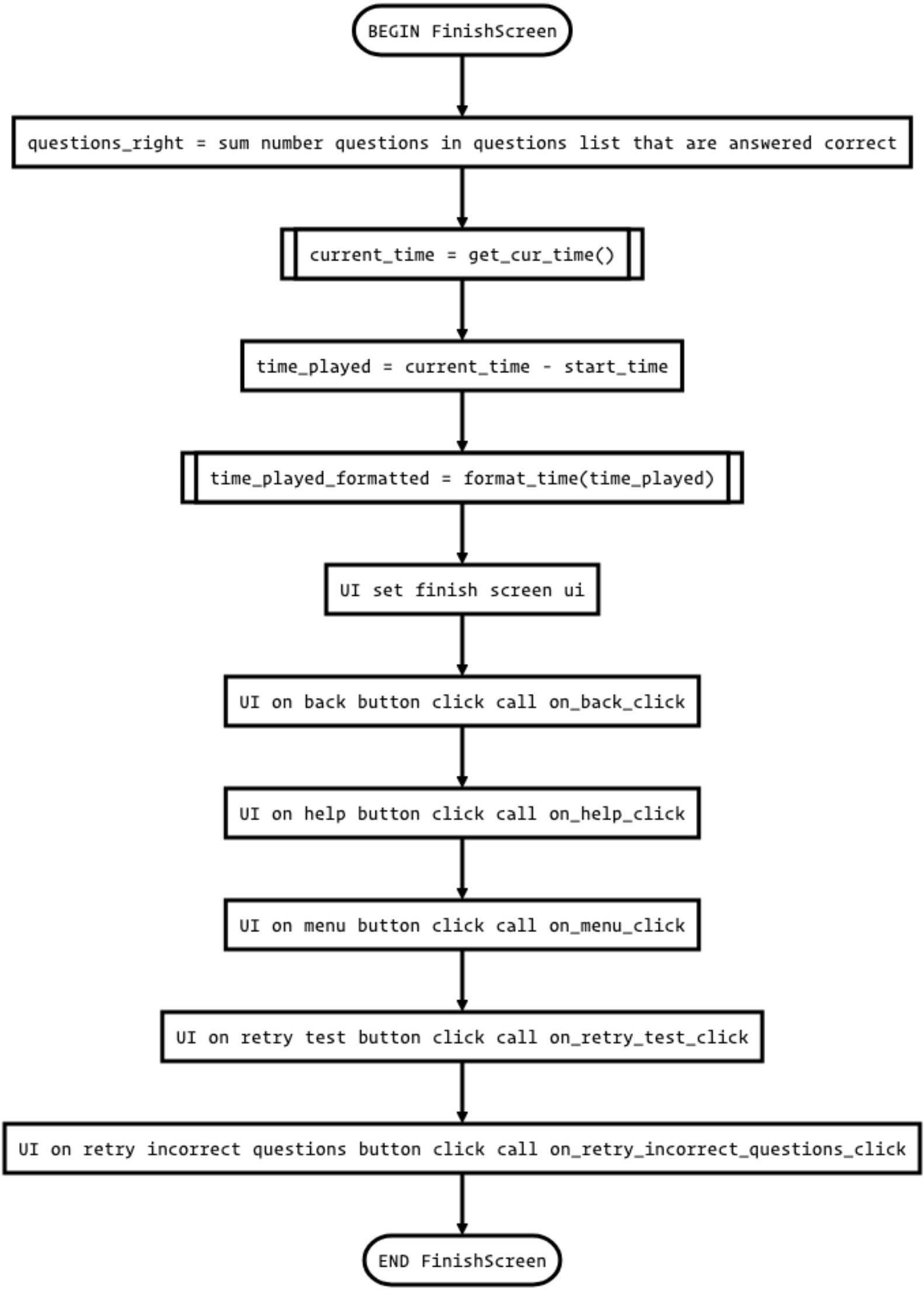

BEGIN num_to_str
result = format num to {dp} decimal places
RETURN result
END num_to_str

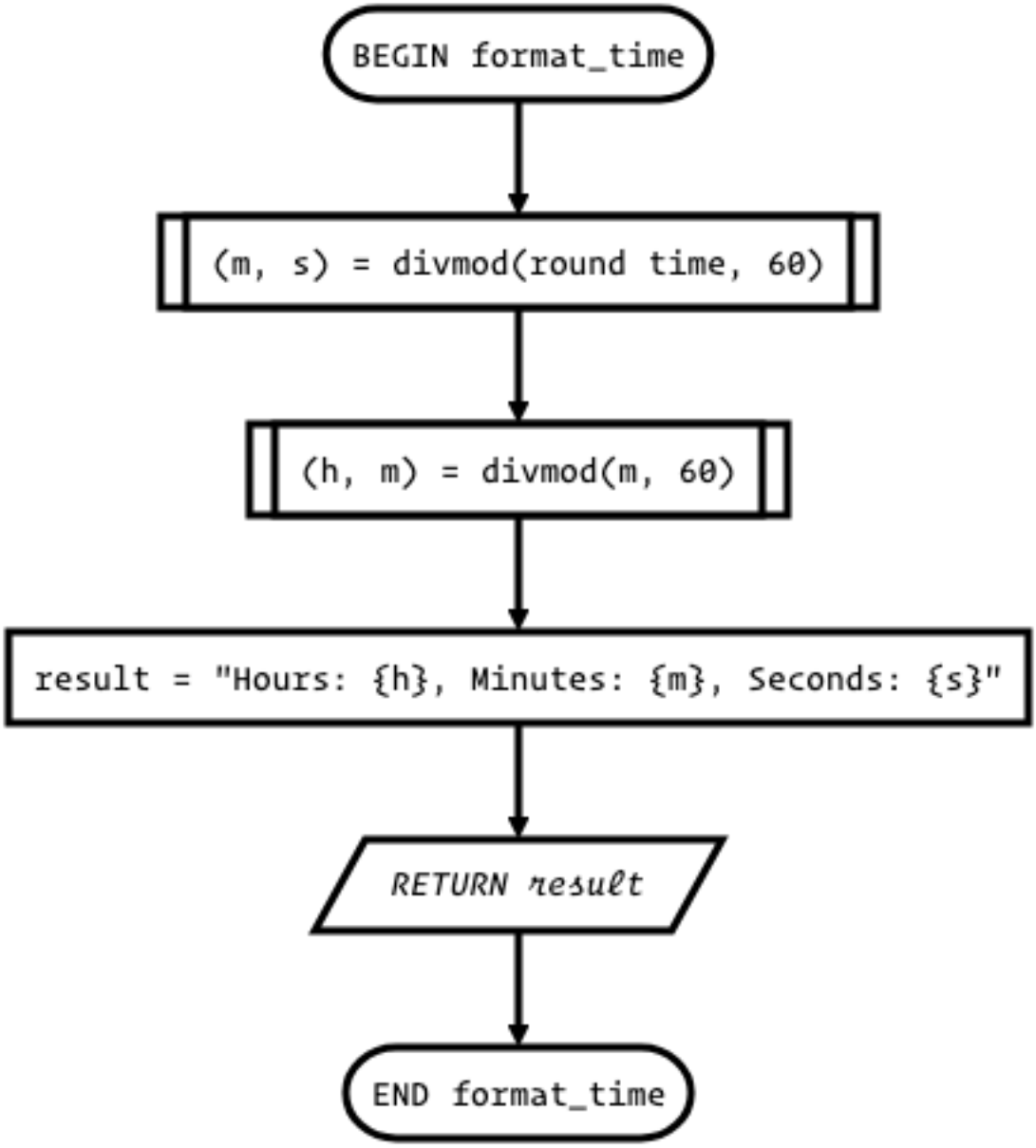

BEGIN get_input_error_msg
float_err = get_float_error_msg(text)
IF float_err
no
yes
RETURN float_err
num = text as float
IF num < 0
no
yes
RETURN "Please enter a positive integer."
RETURN None
END get_input_error_msg

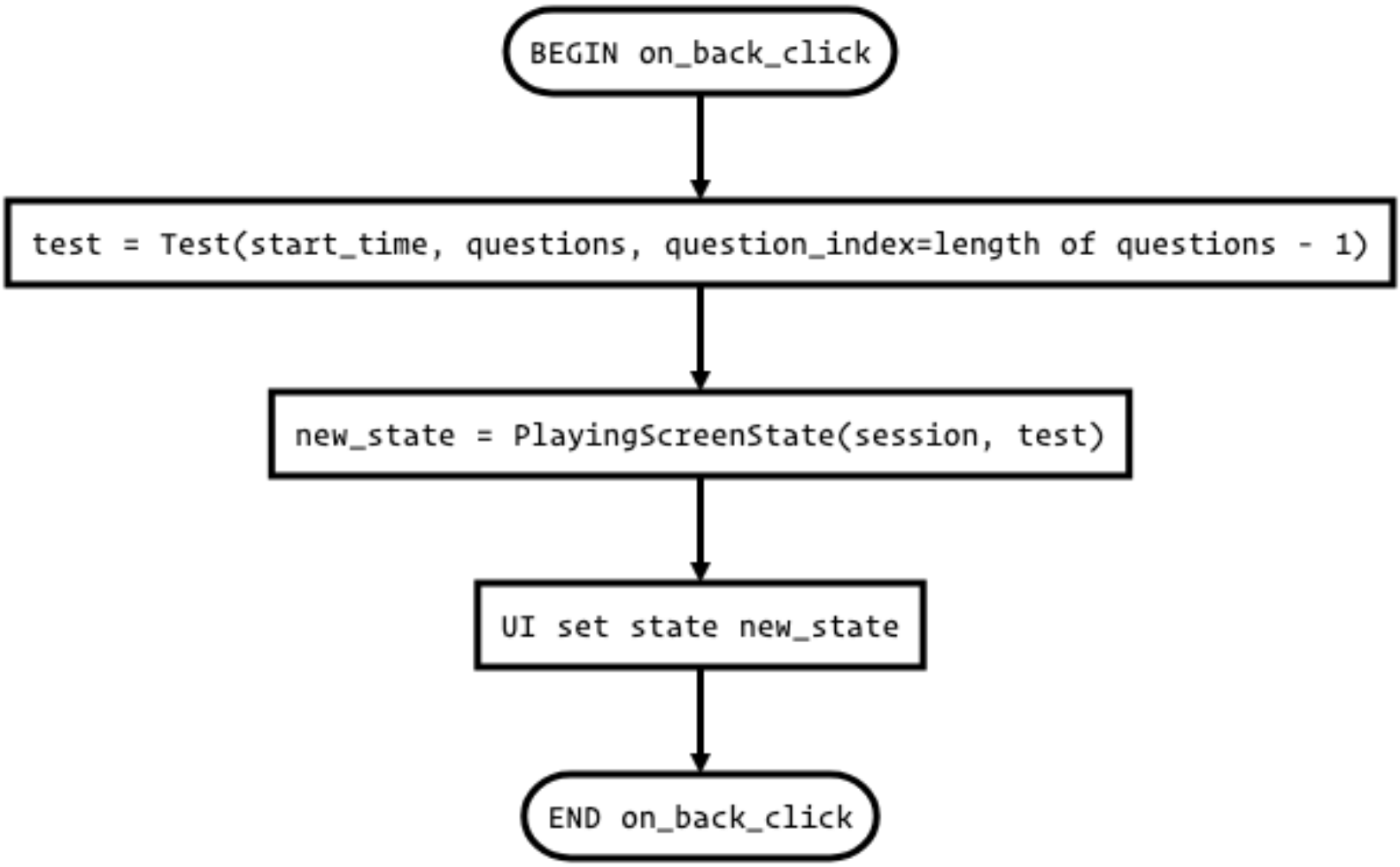

BEGIN input_q_is_ans_correct
correct_ans_as_str = num_to_str(correct_ans)
result = num == correct_ans_as_str
RETURN result
END input_q_is_ans_correct

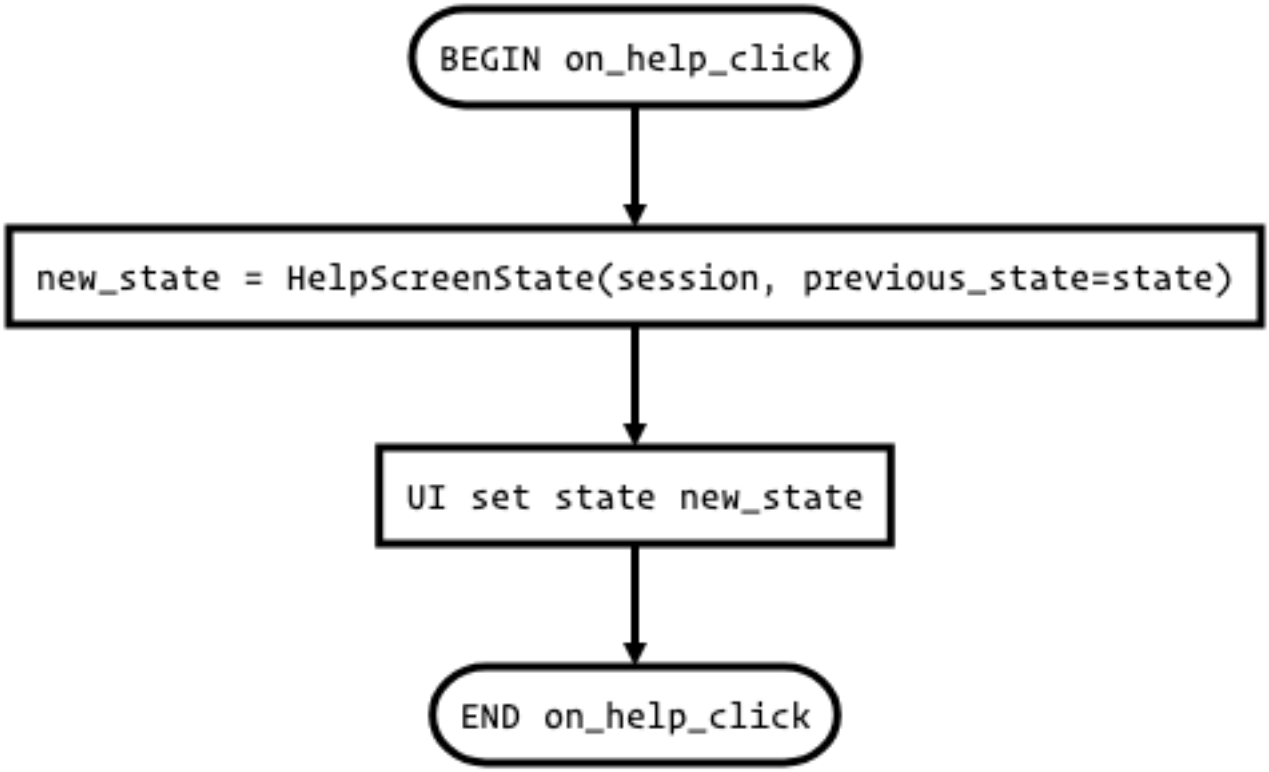

BEGIN make_questions
IF number theory in content
no
yes
add q_bodmas to possible_questions list
IF algebra in content
no
yes
add q_factorise_quadratic to possible_questions list
add q_simplify_linear to possible_questions list
IF geometry in content
no
yes
add q_find_hypot to possible questions list
add q_general_geometry to possible questions list
result = list with length {question_count}, i'th item = make_question(i)
RETURN result
END make_questions

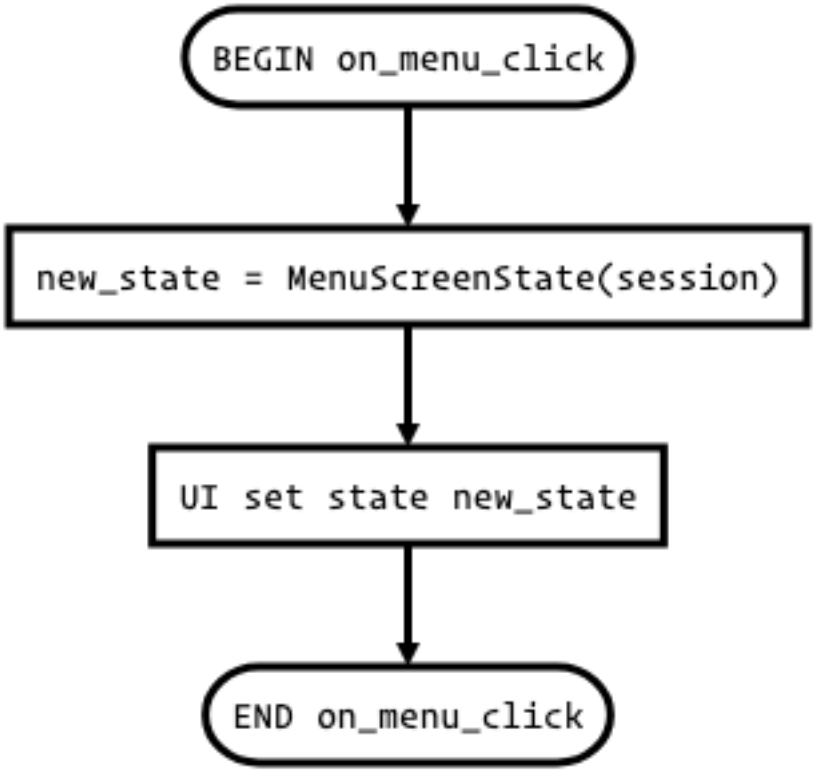

BEGIN make_question
make_question_component = random choice from possible_questions
question_component = make_question_component(controller, question_index)
question = TestQuestion(question_component, answer_state=not answered)
RETURN question
END make_question

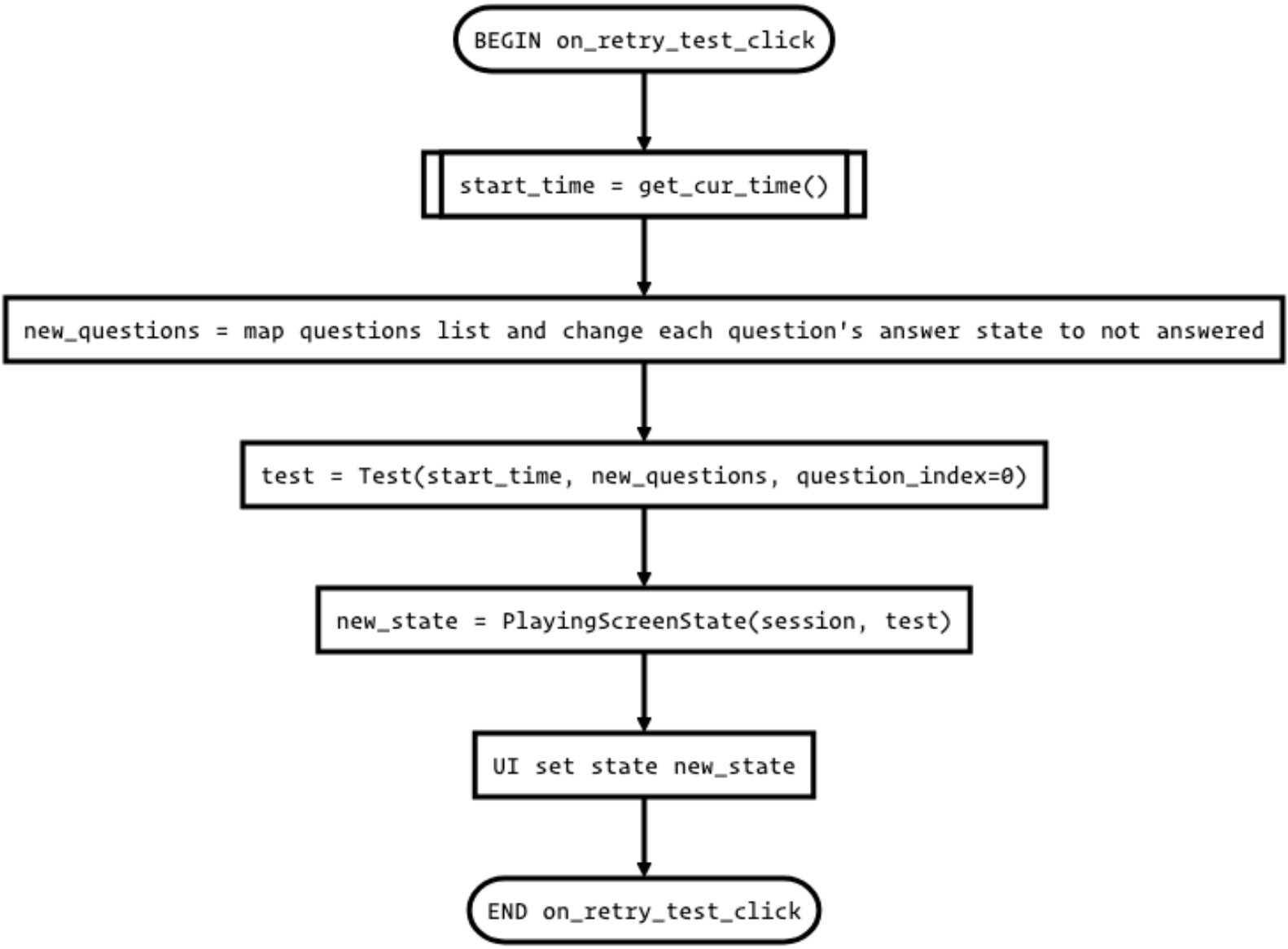

BEGIN FinishScreen
questions_right = sum number questions in questions list that are answered correct
current_time = get_cur_time()
time_played = current_time - start_time
time_played_formatted = format_time(time_played)
UI set finish screen ui
UI on back button click call on_back_click
UI on help button click call on_help_click
UI on menu button click call on_menu_click
UI on retry test button click call on_retry_test_click
UI on retry incorrect questions button click call on_retry_incorrect_questions_click
END FinishScreen

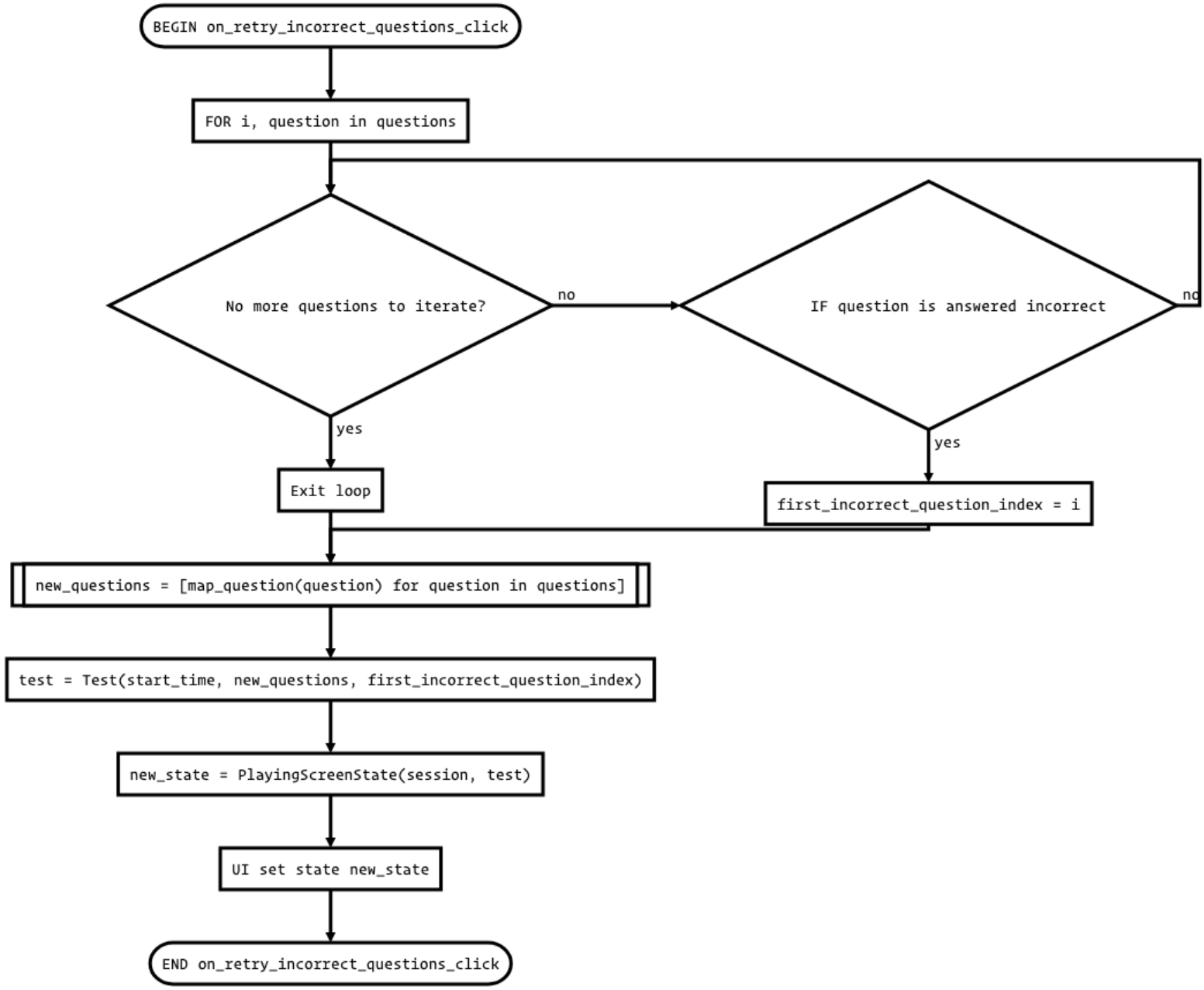

BEGIN format_time
(m, s) = divmod(round time, 60)
(h, m) = divmod(m, 60)
result = "Hours: {h}, Minutes: {m}, Seconds: {s}"
RETURN result
END format_time

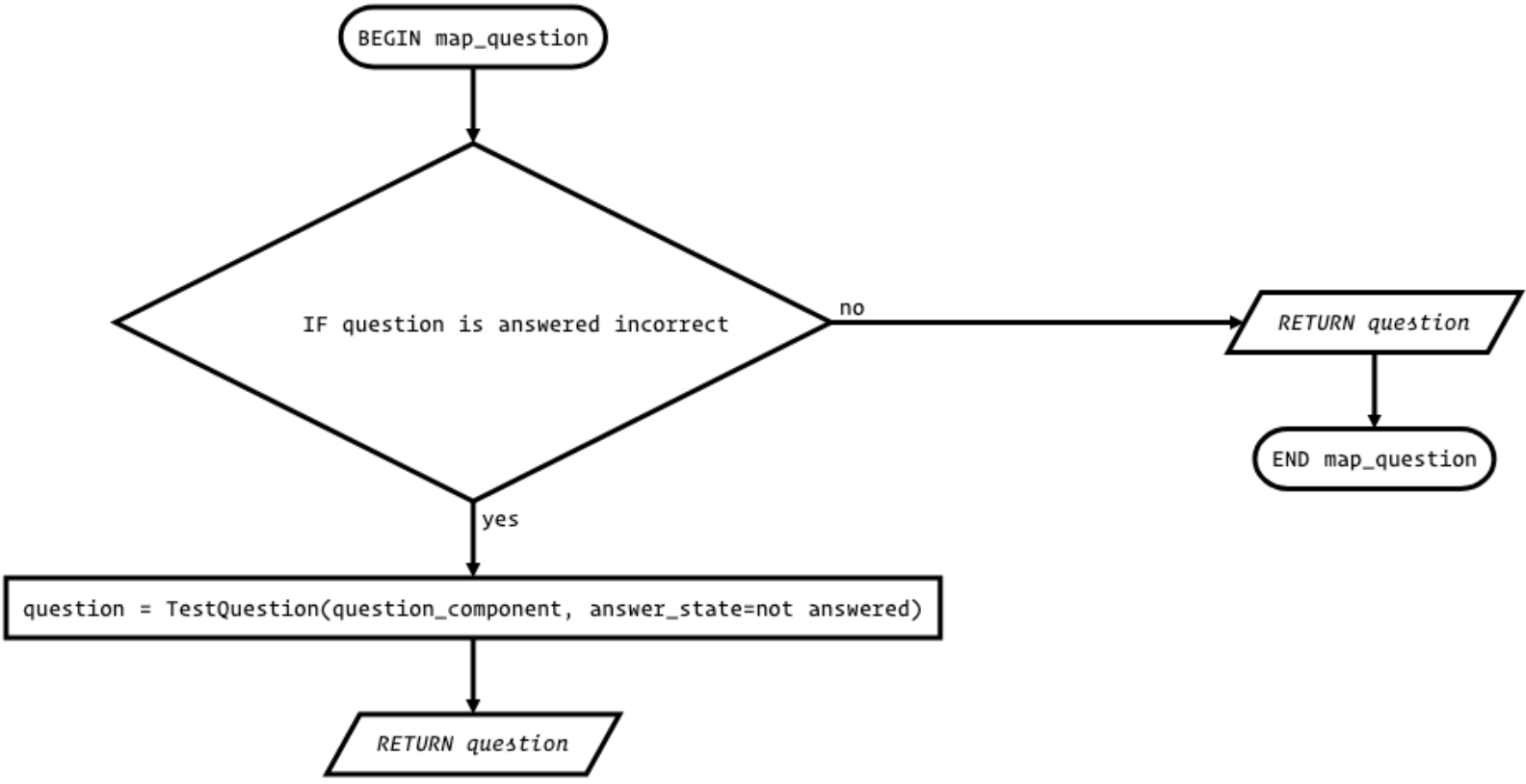

BEGIN on_back_click
test = Test(start_time, questions, question_index=length of questions - 1)
new_state = PlayingScreenState(session, test)
UI set state new_state
END on_back_click

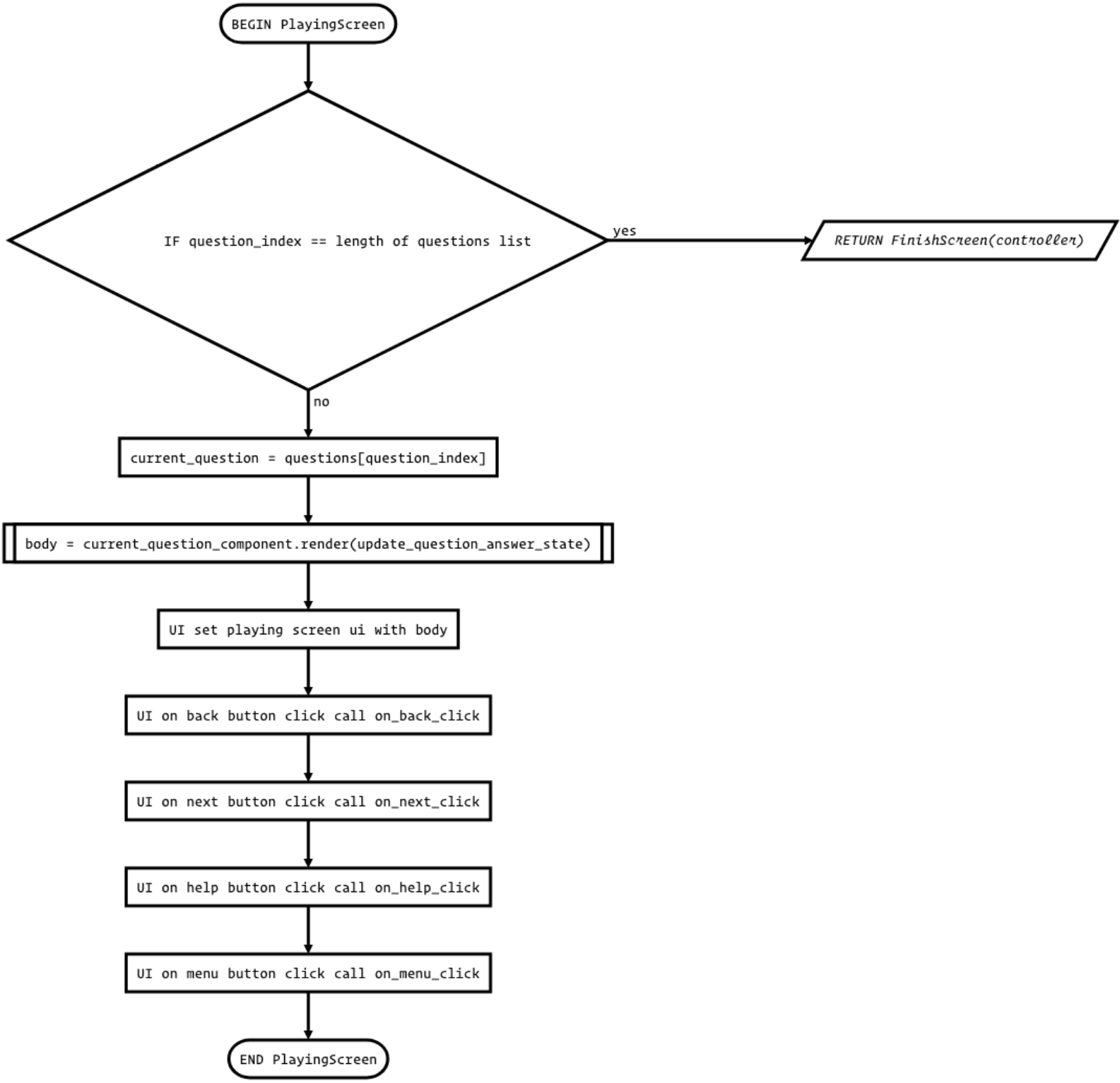

BEGIN on_help_click
new_state = HelpScreenState(session, previous_state=state)
UI set state new_state
END on_help_click

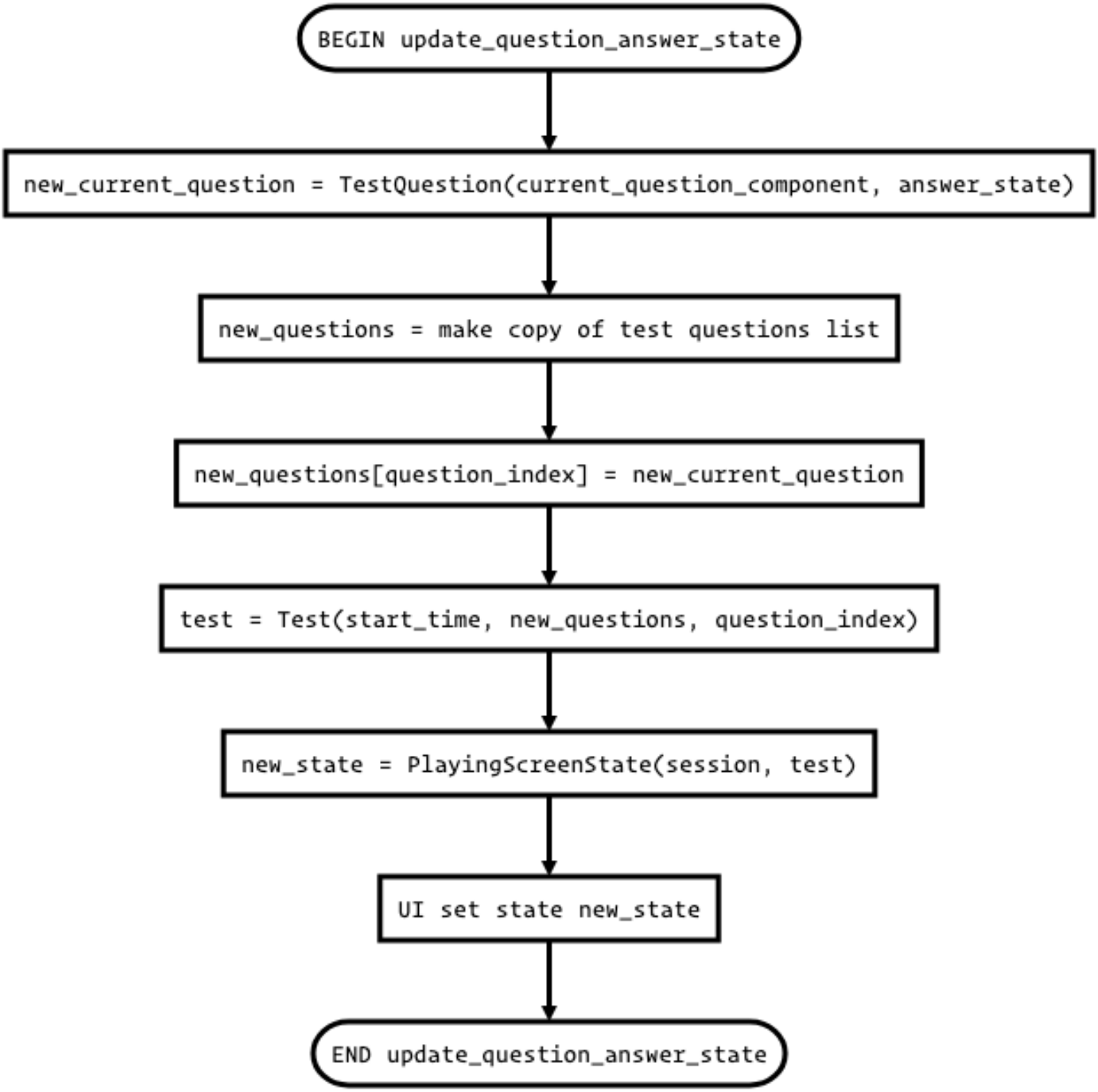

BEGIN on_menu_click
new_state = MenuScreenState(session)
UI set state new_state
END on_menu_click

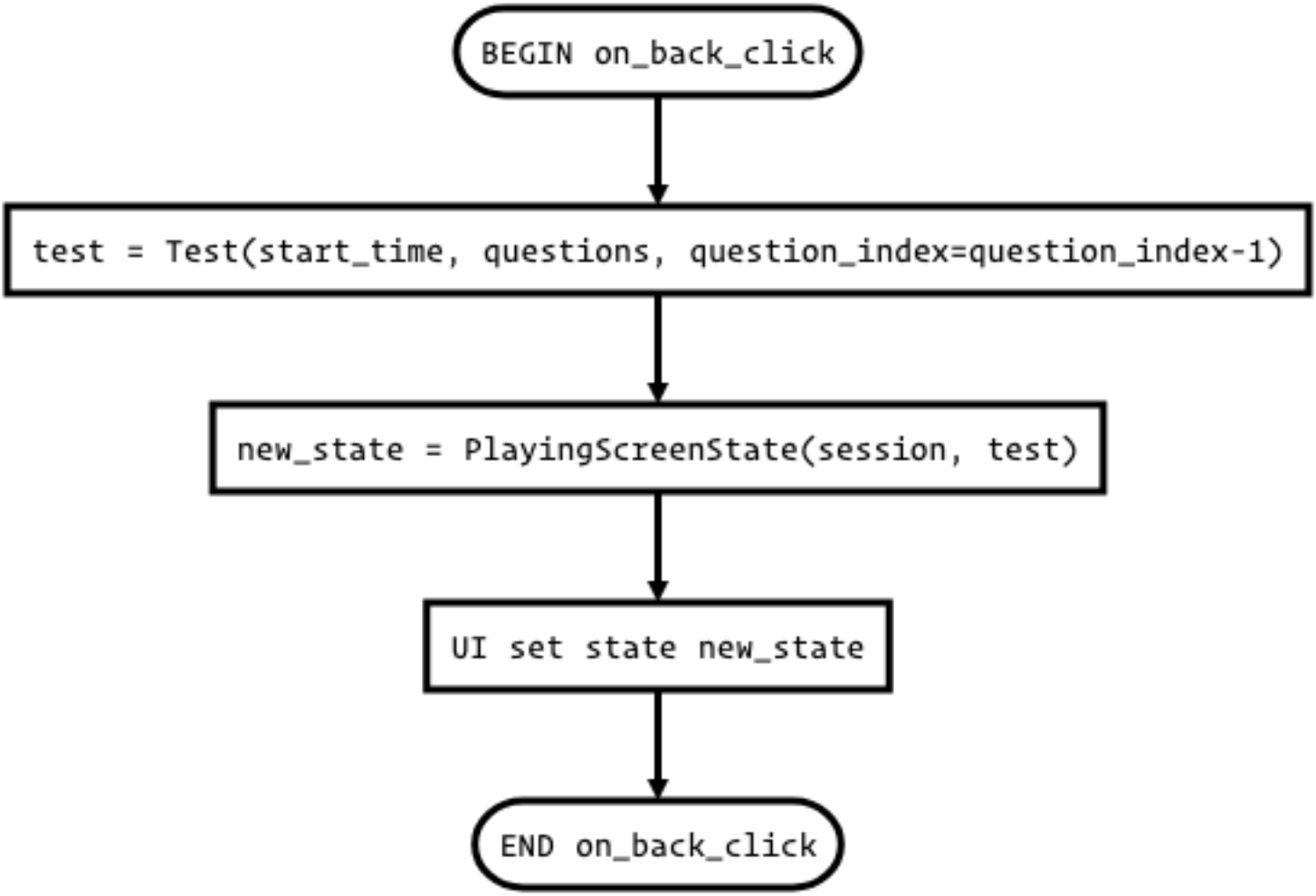

BEGIN on_retry_test_click
start_time = get_cur_time()
new_questions = map questions list and change each question's answer state to not answered
test = Test(start_time, new_questions, question_index=0)
new_state = PlayingScreenState(session, test)
UI set state new_state
END on_retry_test_click

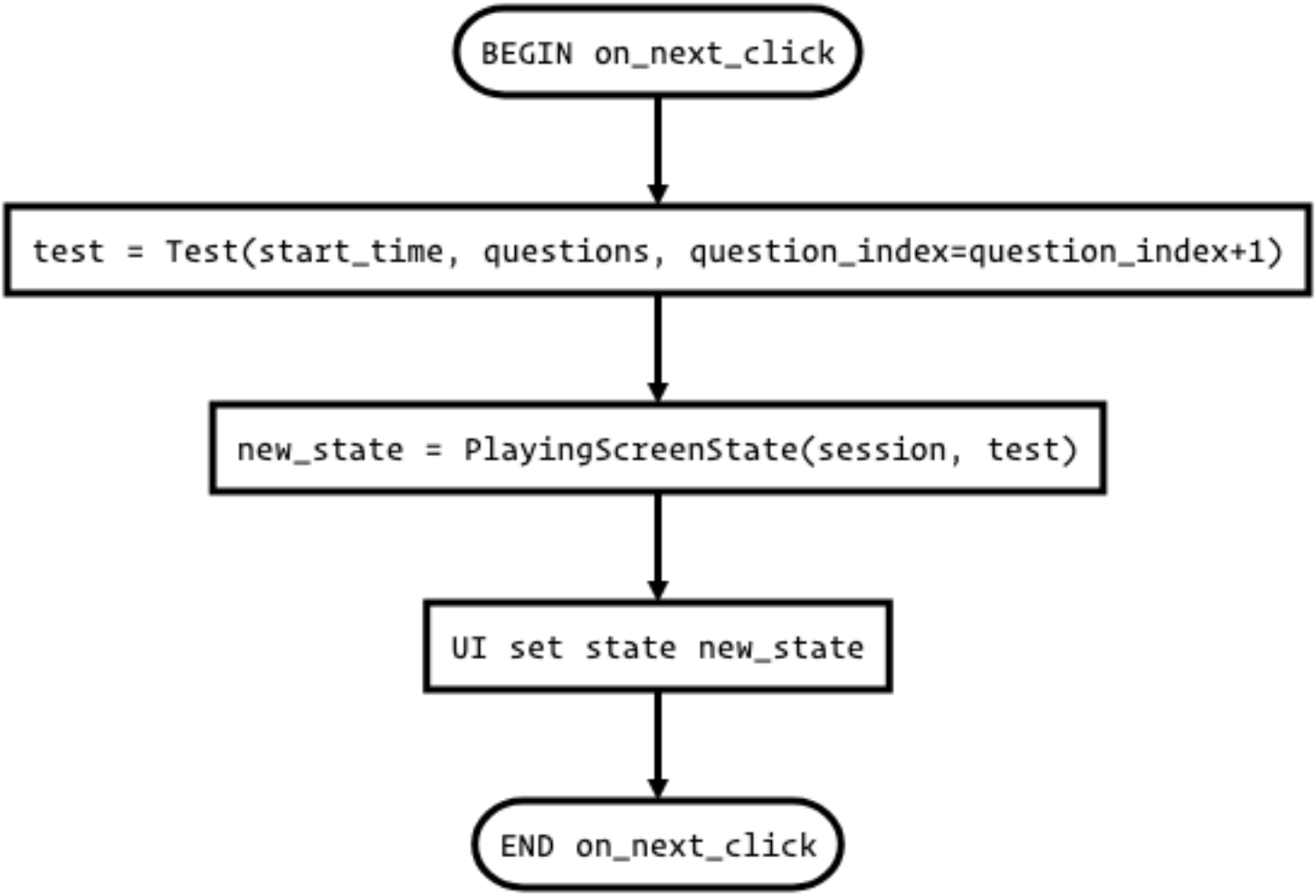

BEGIN on_retry_incorrect_questions_click
FOR i, question in questions
No more questions to iterate?
no
IF question is answered incorrect
no
yes
yes
Exit loop
first_incorrect_question_index = i
new_questions = [map_question(question) for question in questions]
test = Test(start_time, new_questions, first_incorrect_question_index)
new_state = PlayingScreenState(session, test)
UI set state new_state
END on_retry_incorrect_questions_click

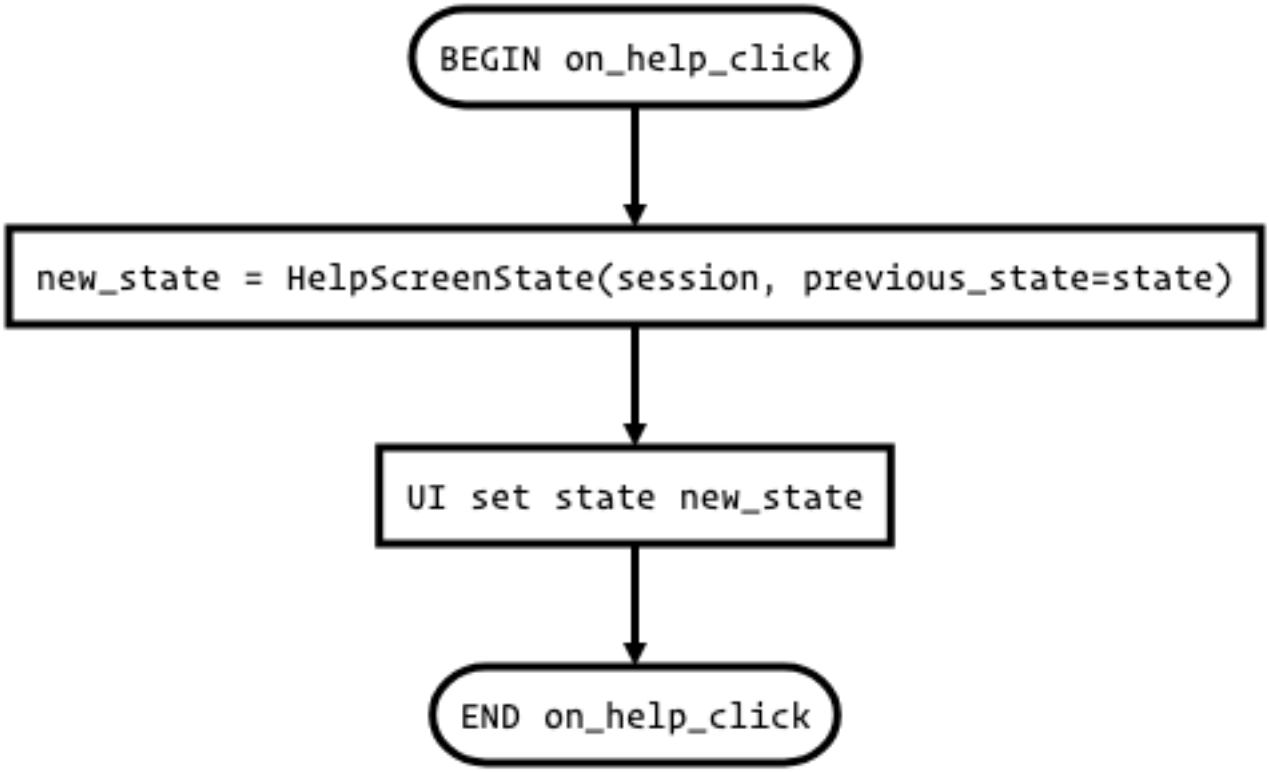

BEGIN map_question
IF question is answered incorrect
no
RETURN question
END map_question
yes
question = TestQuestion(question_component, answer_state=not answered)
RETURN question

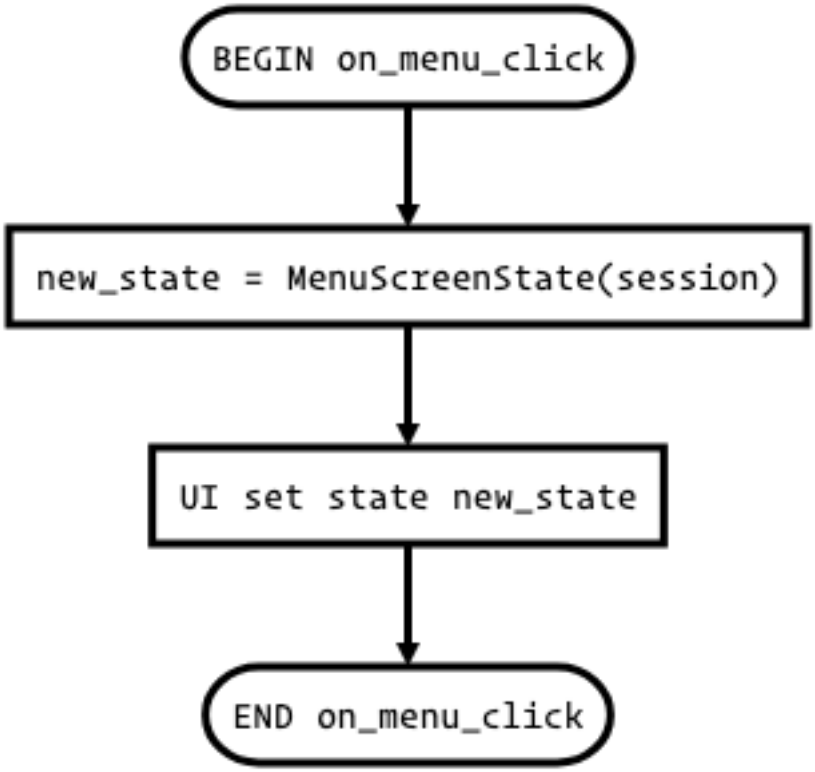

BEGIN PlayingScreen
IF question_index == length of questions list
yes
RETURN FinishScreen(controller)
no
current_question = questions[question_index]
body = current_question_component.render(update_question_answer_state)
UI set playing screen ui with body
UI on back button click call on_back_click
UI on next button click call on_next_click
UI on help button click call on_help_click
UI on menu button click call on_menu_click
END PlayingScreen

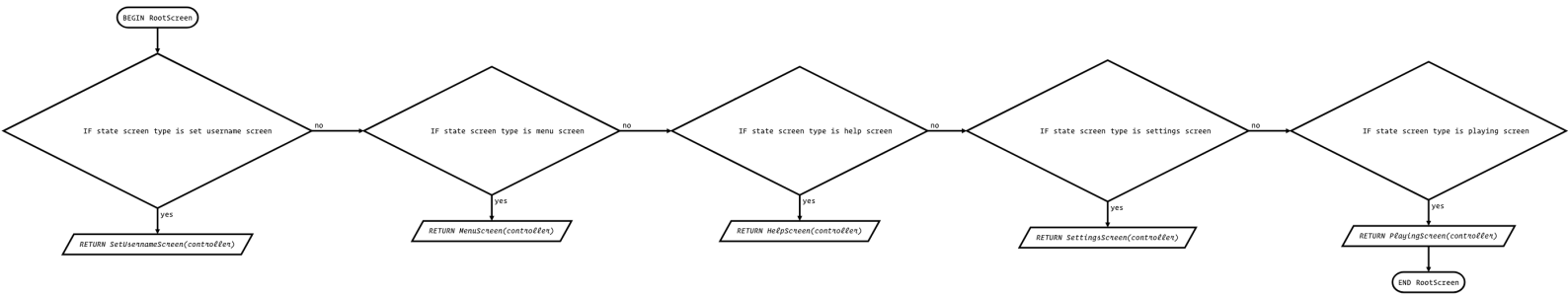

BEGIN update_question_answer_state
new_current_question = TestQuestion(current_question_component, answer_state)
new_questions = make copy of test questions list
new_questions[question_index] = new_current_question
test = Test(start_time, new_questions, question_index)
new_state = PlayingScreenState(session, test)
UI set state new_state
END update_question_answer_state


BEGIN on_back_click
test = Test(start_time, questions, question_index=question_index-1)
new_state = PlayingScreenState(session, test)
UI set state new_state
END on_back_click


BEGIN on_next_click
test = Test(start_time, questions, question_index=question_index+1)
new_state = PlayingScreenState(session, test)
UI set state new_state
END on_next_click


BEGIN on_help_click
new_state = HelpScreenState(session, previous_state=state)
UI set state new_state
END on_help_click


BEGIN on_menu_click
new_state = MenuScreenState(session)
UI set state new_state
END on_menu_click


BEGIN RootScreen
IF state screen type is set username screen
no
yes
RETURN SetUsernameScreen(controller)
IF state screen type is menu screen
no
yes
RETURN MenuScreen(controller)
IF state screen type is help screen
no
yes
RETURN HelpScreen(controller)
IF state screen type is settings screen
no
yes
RETURN SettingsScreen(controller)
IF state screen type is playing screen
yes
RETURN PlayingScreen(controller)
END RootScreen