SyntaxError: dziubkiem pokazuje, gdzie znajduje się błąd

**pass statements**

https://docs.python.org/3/tutorial/controlflow.html

https://realpython.com/python-pass/

**INTRODUCTION**

'pass' is a keyword and an entire statement. This statement doesn’t do anything. It is used when a statement is required syntactically but you do not want any code to execute.

Use cases:   
(a) future code,   
(b) planning branching logic (if-elif-else),   
(c) commenting out code,   
(d) breakpoints for debuggers,   
(e) new exceptions, catching exceptions.

while True: # infinite loop

pass # waiting for keyboard interrupt

def dummy\_function():

pass # to complete later

class EmptyClass:

pass # to complete later

class InvalidPasswordError(ValueError): # a name is important only

pass

if x % 2 == 0: # two cases to consider

pass # what to do if x is even

else:

pass # what to do if x is odd

if condition:

#statement1 # commented out in debugging, because it takes a long time

pass

else:

statement2

**Expression statements**

https://docs.python.org/3/reference/expressions.html

https://stackoverflow.com/questions/4728073/what-is-the-difference-between-an-expression-and-a-statement-in-python

**INTRODUCTION**

All Python expressions can be used as statements. Expressions can be reduced to some kind of 'value', but in statements we use 'side effects' (function calls). Expression statements are often used in the interactive mode.

Wyrażenia można gromadzić w liście, instrukcji – nie można

# Expression statements.

2 + 5 # arithmetic or boolean operations

name # variable

alist[idx] # with lists, tuples, dicts

[x for x in iterable] # list|dict|set comprehension

some\_function() # function call

some\_object.method() # method call

print(word) # function in Py3 (the result is None and a message)

yield x \*\* 2 # in generators

lambda x: x \*\* 2 # anonymous function (lambda function)

A if condition else B # conditional expression

# Statements, but not expressions.

print word # statement in Py2

a = 5 # assignment

return name

if condition: pass

def some\_function(): pass

class FirstClass: pass

**if statements**

https://docs.python.org/3/tutorial/controlflow.html

**INTRODUCTION**

The 'if' statement is used for conditional execution. It is a compound statement.

# Compound statement structure.

header\_statement: # 'if' or 'while' or 'for'

statements # indentation 4 spaces (recommended)

# Nested suites in compound statements.

suite1 # no indentation

header\_statement1:

suite2 # indentation 4 spaces

header\_statement2:

suite3 # total indentation 8 spaces

suite2 (continued) # indentation 4 spaces

suite1 (continued) # no indentation

# Syntax. Zero or one part of the 'if' statement will be executed.

Pierwszy napotkany spełniony warunek będzie wykonany, to ma znaczenie jaka jest kolejność warunków

if condition1:

statements

elif condition2: # optional, more 'elif' parts is possible

statements

else: # optional, only one 'else' part is possible

statements

# Short form in a single line (better to avoid).

if condition: simple\_statement

assert isinstance(n, int)

if n < 0:

print("negative number")

elif n % 2 == 0:

print("{} {}".format(n, "is even"))

else:

print("{} {}".format(n, "is odd"))

**CONDITIONAL EXPRESSION**

if condition:

A = X

else:

A = Y

# Conditional expression (sometimes called a “ternary operator”) [PEP 308].

A = X if condition else Y

A = ((X) if (condition) else (Y)) # for more complex expressions

# In C/C++:

# A = ((condition) ? (X) : (Y))

a, b = 25, 30

pos\_a = ((-a) if a < 0 else a) # abs(a) is better

max\_ab = (b if a < b else a) # max(a, b) is better

min\_ab = (a if a < b else b) # min(a, b) is better

**match statements**

https://docs.python.org/3/tutorial/controlflow.html#match-statements

https://docs.python.org/3/reference/compound\_stmts.html#match

PEP 634 - Structural Pattern Matching: Specification

PEP 636 - Structural Pattern Matching: Tutorial

**INTRODUCTION**

A 'match' statement takes an expression and compares its value to successive patterns given as one or more case blocks. This statement was introduced in Python 3.10.

# The simplest form of a 'match' statement.

Odpowiednik “Which” w Mathematica

def http\_error(status, flag=False):

match status: # header line

case 400:

return "Bad request"

case 401 | 403 | 404: # several literals can be combined using |

return "Not allowed"

case 404:

return "Not found"

case 418 if flag: # using a guard

return "I'm a teapot"

case \_: # optional, never fails to match

return "Something's wrong with the internet"

**while loops**

https://docs.python.org/3/reference/compound\_stmts.html#while

**INTRODUCTION**

The 'while' statement is used for repeated execution as long as an expression is true.

# Syntax.

while condition1: #może być wykonany zero razy

statements

if condition2:

break # optional, terminate the loop without executing 'else'

if condition3:

continue # optional, go to condition1

statements

else: # optional

statements # executed if condition1 is false, the loop terminates

n = 20

while n > 0:

if n == 13:

n = n - 1

continue # skipping 13

print(n)

if (n % 2) == 0:

n = n // 2

else:

n = n - 1

while True: # infinite loop gdy nie wiemy ile razy mamy wykonać

#reply = raw\_input("Input word (or 'stop'):") # Py2

reply = input("Input word (or 'stop'):") # Py3

if reply == "stop":

break

print(reply.upper())

**for loops**

https://docs.python.org/3/tutorial/controlflow.html

**INTRODUCTION**

The 'for' statement is used to iterate over the elements of an iterable object.

# Syntax.

for item in iterable: #lista, krotka, znaki w stringu

statements # item is processed

if condition1:

break # optional, terminate the loop without executing 'else'

if condition2:

continue # optional, go to the next item from iterable

statements

else: # optional, terminate the loop

statements # executed if 'break' was not used, even for an empty iterable

# Warning: the name 'item' is created only if 'iterable' is not empty!

# Range-based for loop in C++ (since C++11)

# for (auto item : container) statement

# for (auto& item : container) statement

word = "Python"

for char in word: # use decriptive names

if char == "o":

break # characters printed: 'P', 'y', 't', 'h'

print(char)

assert char == "o" # 'char' is alive, można przypisać

fruits = ["apple", "pear", "plum", "apricot"] # use plural when possible

for fruit in fruits:

if fruit[0] == "a":

continue # skipping "apple" and "apricot"

print(fruit)

assert fruit == "apricot"

# What if we need an index?

for i in range(len(fruits)): # C style

print("{} {}".format(i, fruits[i]))

for (i, fruit) in enumerate(fruits): # recommended

print("{} {}".format(i, fruit))

[c\*i for (i, c) in enumerate("abcd")] # ["", "b", "cc", "ddd"]

# loop over pairs

for t in [(1. "one"), (2, "two"), (3, "three")]:

print("first {} second {}".format(t[0], t[1]))

# better version with unpacking

for (a, b) in [(1. "one"), (2, "two"), (3, "three")]:

print("first {} second {}".format(a, b))

# Code that modifies a collection while iterating over that same collection

# can be tricky to get right.

# It is safe to extend the list (append at the end).

# It is not safe to add or remove an item in the middle of the list.

L = [0, 1, 2, 3, 4, 5]

for i in L:

print(i)

if (i % 2) == 1: # if i is odd then extend the list

L.append(2 \* i) # adding two even numbers

L.append(4 \* i)

L = [0, 1, 2, 3, 4, 5]

for i in L[:]: # use copy in the loop [or list(L)]

if (i % 2) == 1:

L.remove(i) # changing the original list

# Better solution (filtering).

new\_list = [i for i in L if (i % 2) != 1]

# Looping over a sequence in reverse.

for char in reversed("qwerty"): # 'reversed object'

print(char) # y, t, r, e, w, q

# Looping over a sequence in sorted order.

for char in sorted("qwerty"): # ['e', 'q', 'r', 't', 'w', 'y']

print(char)

# Looping over two or more sequences at the same time.

letters = ["a", "b", "c", "d"]

numbers = [10, 20, 30, 40]

for (letter, number) in zip(letters, numbers):

print("{} {}".format(letter, number))

# Eliminating duplicate elements.

for char in set("abracadabra"): # 'a', 'c', 'd', 'b', 'r' zbiór

print(char)

# Using all() and any().

L = [1, 2, 4, 5]

if all(x % 2 == 0 for x in L): #zqraca true gdy wszystkie warunki są true

print("all numbers are even")

if any(x % 2 == 0 for x in L):

print("there is an even number on the list")