Adrien Protzel

1.1: BNF

<S> := “begin” <stmts> “end”  
<stmts> := <statement> ; <stmts>| <empty>

1.2: Ambiguous Grammars

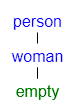
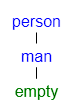
Example Strings:

<person> ::= <woman> | <man>

<woman> ::= wilma | betty | <empty>

<man> ::= fred | barney | <empty>

Example Trees:



Since <empty> can be reached by two distinctly different trees, it is ambiguous.  
It should move the <empty> to the <person> list to make it not ambiguous.

1.3: Python Grammar

1.3: Code:

# Adrien Protzel

import random

class Helldives:

def \_\_init\_\_(self, bugs, bots):

self.bugs = bugs

self.bots = bots

def render(self): # stub

pass

Missions = []

for \_ in range(random.randint(10+1, 20-1)):

Missions.append(Helldives)

print(f"There are {len(Missions)} number of objects in the list.")

1.3: Parse Tree: <only class definition>

ClassDef

┝ name: Helldives

┝ bases: []

┝ keywords: []

┕ body:

┝ FunctionDef

⏐ ┝ name: \_\_init\_\_

⏐ ┝ args:

⏐ ⏐ ┕ arguments

⏐ ⏐ ┝ posonlyargs: []

⏐ ⏐ ┝ args:

⏐ ⏐ ⏐ ┝ arg: self

⏐ ⏐ ⏐ ┝ arg: bugs

⏐ ⏐ ⏐ ┕ arg: bots

⏐ ⏐ ┝ vararg: None

⏐ ⏐ ┝ kwonlyargs: []

⏐ ⏐ ┝ kw\_defaults: []

⏐ ⏐ ┝ kwarg: None

⏐ ⏐ ┕ defaults: []

⏐ ┕ body:

⏐ ┝ Assign

⏐ ⏐ ┝ targets:

⏐ ⏐ ⏐ ┕ Attribute

⏐ ⏐ ⏐ ┝ value: Name(self)

⏐ ⏐ ⏐ ┕ attr: bugs

⏐ ⏐ ┕ value: Name(bugs)

⏐ ┕ Assign

⏐ ┝ targets:

⏐ ⏐ ┕ Attribute

⏐ ⏐ ┝ value: Name(self)

⏐ ⏐ ┕ attr: bots

⏐ ┕ value: Name(bots)

┕ FunctionDef

┝ name: render

┝ args:

⏐ ┕ arguments

⏐ ┝ posonlyargs: []

⏐ ┝ args:

⏐ ⏐ ┕ arg: self

⏐ ┝ vararg: None

⏐ ┝ kwonlyargs: []

⏐ ┝ kw\_defaults: []

⏐ ┝ kwarg: None

⏐ ┕ defaults: []

┕ body:

┕ Pass

1.3: AST Tree:

(Let it be known that this answer is a bit of a cheat, I found an api to automatically spit out an AST tree from a string of code. The program to do this will be attached in a zip file)

Module(

body=[

Import(

names=[

alias(name='random')]),

ClassDef(

name='Helldives',

bases=[],

keywords=[],

body=[

FunctionDef(

name='\_\_init\_\_',

args=arguments(

posonlyargs=[],

args=[

arg(arg='self'),

arg(arg='bugs'),

arg(arg='bots')],

kwonlyargs=[],

kw\_defaults=[],

defaults=[]),

body=[

Assign(

targets=[

Attribute(

value=Name(id='self', ctx=Load()),

attr='bugs',

ctx=Store())],

value=Name(id='bugs', ctx=Load())),

Assign(

targets=[

Attribute(

value=Name(id='self', ctx=Load()),

attr='bots',

ctx=Store())],

value=Name(id='bots', ctx=Load()))],

decorator\_list=[],

type\_params=[]),

FunctionDef(

name='render',

args=arguments(

posonlyargs=[],

args=[

arg(arg='self')],

kwonlyargs=[],

kw\_defaults=[],

defaults=[]),

body=[

Pass()],

decorator\_list=[],

type\_params=[])],

decorator\_list=[],

type\_params=[]),

Assign(

targets=[

Name(id='Missions', ctx=Store())],

value=List(elts=[], ctx=Load())),

For(

target=Name(id='\_', ctx=Store()),

iter=Call(

func=Name(id='range', ctx=Load()),

args=[

Call(

func=Attribute(

value=Name(id='random', ctx=Load()),

attr='randint',

ctx=Load()),

args=[

BinOp(

left=Constant(value=10),

op=Add(),

right=Constant(value=1)),

BinOp(

left=Constant(value=20),

op=Sub(),

right=Constant(value=1))],

keywords=[])],

keywords=[]),

body=[

Expr(

value=Call(

func=Attribute(

value=Name(id='Missions', ctx=Load()),

attr='append',

ctx=Load()),

args=[

Name(id='Helldives', ctx=Load())],

keywords=[]))],

orelse=[]),

Expr(

value=Call(

func=Name(id='print', ctx=Load()),

args=[

JoinedStr(

values=[

Constant(value='There are '),

FormattedValue(

value=Call(

func=Name(id='len', ctx=Load()),

args=[

Name(id='Missions', ctx=Load())],

keywords=[]),

conversion=-1),

Constant(value=' number of objects in the list.')])],

keywords=[]))],

type\_ignores=[])

1.4: Python Grammar II

Equally simple and helpful statements could be “wait” and “skip”.

Wait: It holds the program for a set time, perhaps to make a program take longer for an output to catch up or to give an illusion of loading. For example, “wait 5” could pause the execution for 5 seconds.

Skip: This keyword skips the next block of code, which can be invaluable for debugging. For instance, skip could be used within loops or conditional statements to bypass certain conditions without breaking the loop or exiting the function.