Santa

Simple and New

Technical Articulation

Summary and Example Programs

Version 12.25

Jenna Ficula

This is the documentation for a new festive programming language, Santa, for a language project in Theory of Programming Languages at Marist College.

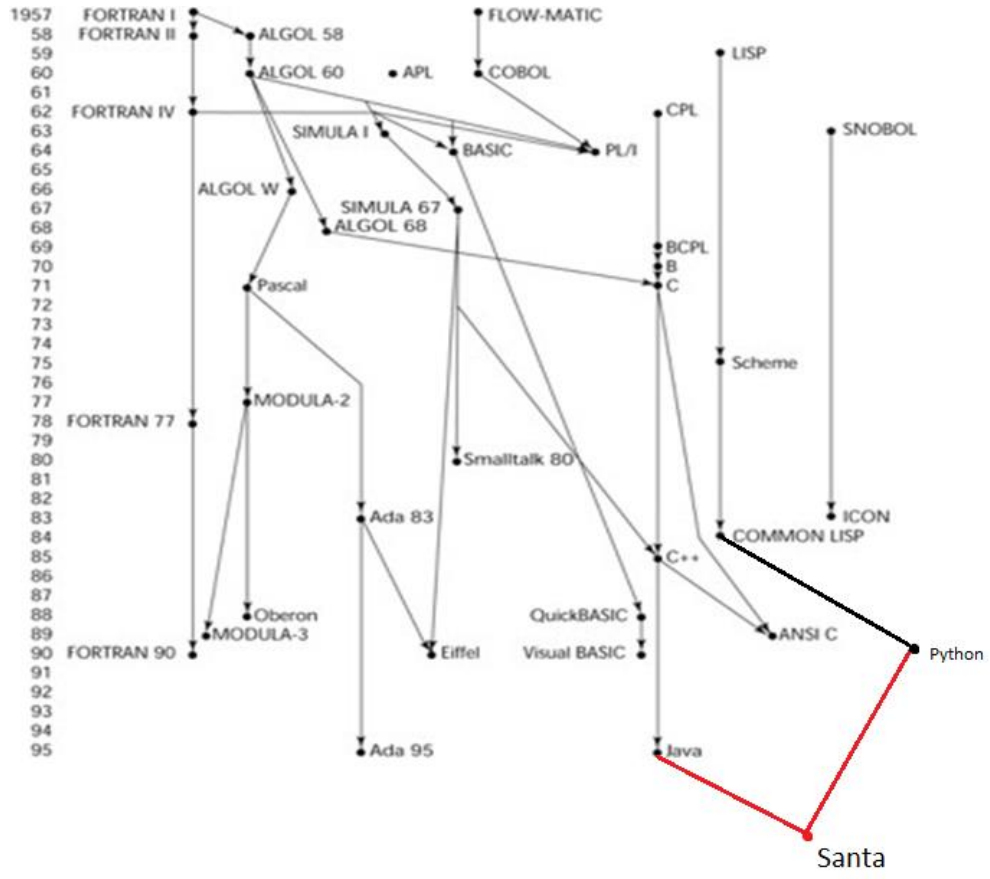
1. Introduction

Santa or (Simple and New Technical Articulation) is a modern strongly-typed, object-oriented language best used during the winter season in a procedural environment. It is based on Python, Java, and Christmas Spirit, but differing in the following ways:

1. Brackets have been removed and instead Merry and Christmas determine when each function or class initiates and stops.
2. Santa is strongly typed so all variables must be one of the included data types
3. Santa is statically scoped and utilizes early binding.
4. Everything is termed in the manner of the Holiday Season.
5. Santa is compiled.
6. There are no Null values in Santa, only Coal.
7. Return values are received. (ie. receive Coal) if a value is null.
8. Hohoho is print

1.1 Genealogy

The origins of Santa and how this language fits into the programming language genealogy. Python initially originated from LISP, Common LISP and from features from functional programming languages such as SETL and Haskell with some combined type features of C. Java also is derived from the origins of the programing language C. Therefore, Santa sits at the bottom between the Java and Python.



1.2 Hello world

Morally obligated to write the "Hello World" program Santa.

```
1. def helloWorld()
2. Merry
3.          hohoho("Hello World!")
4. Christmas
```

1.3 Program structure

The most important organizational concepts in Santa are as follows

1. Every elf (class) is public to spread Christmas cheer.
2. Elves (classes) must be inside a workshop (similar to Java packages). Workshops can have multiple elves and this is a mechanism to encapsulate a group of elves (classes) so that different workshops can have elves of the same name.
1. Elf (class) attributes are get and set with the phrase elfself.attribute.
2. Santa uses white space like python and can be created with tabs or spaces although the preferred method is spaces.
3. Functions are defined like in Python with def keyword but must have explicitly declared perimeter and return types declared similar to Java.
4. The elf (class) as well as each function, for, and while loop must begin with Merry and end with Christmas rather than brackets.
5. A main function is not required but you can use one.

```
5. workshop NorthPole
6. Merry
7.   elf Child
8.   Merry
9.          def Child(elfself, Carol fname, Carol lname, Num gpa)
10.           Merry
11.             elfself.fname is fname
12.             elfself.lname is name
13.             elfself.gpa is 3.0
14.           Christmas
15.
```

```
16.            def displayGpa(elfself)
17.            Merry
18.                 hohoho ("gpa " + elfself.gpa)
19.            Christmas
20.
21.            def displayChild(elfself)
22.            Merry
23.                  hohoho ("First  ", elfself.fname, "Last  ", elfself.lname)
24.            Christmas
25.
26.           def Boolean isOnNiceList(elfself)
27.           Merry
28.                check if elfself.gpa() > 3.0
29.                   receive nice
30.                else
31.                   receive naughty
32.           Christmas
33.    Christmas
34. Christmas
35. "This would create first object of Child class"
36. newchild = Child("Jenna", “Ficula”, 3.6)
37. newchild.displayChild()
38.  >> First  Jenna Last  Ficula
```

This example declares a ***workshop*** NorthPole (package) and also an ***elf*** (class) named Child. This elf (class) contains three attributes fname, lname, and gpa. It contains a default constructer which takes three arguments which are specified by type ***Carol*** (String) and ***Num*** (Int) as wel as value. The getter and setter methods are created with no programmer input necessary and can be allowed the class attributed to be referred to using ***elfself***. The elf also has methods ***displayGpa*** and ***displayChild*** to print out the gpa and name of child respectively using the ***hohoho*** aka print command. The elf has a method ***isOnNiceList()*** which will return a Boolean (***Naughty*** or ***Nice***) based on the current gpa of the given child. Keywords ***Merry*** and ***Christmas*** additionally frame each function and the entire class declaration.

1.4 Types and variables

There are two kinds of types in Santa ***value types*** and ***reference types***. Variables of value types directly contain their data whereas variables of reference types store references to their data, the latter being known as objects. With reference types, it is possible for two variables to reference the same object and thus possible for operations on one variable to affect the object referenced by the other variable.

1.5 Statements Differeing from Python and Java

Statement	**Example**
Expression statement	def returnValue Main() Merry Num Reindeer Reindeer is 0 Carol child Child is “Jenna” Christmas <<- equality (=) becomes is
Constructor	def myElf (elfself, type value, type value) Merry elfself.attribute1 is “ ” elfself.attribute2 is 0 Christmas
If Statement	def returnValue Main() Merry check if (x > 0) receive value check else if () receive value else receive value Christmas
Define Arrays (allIWantForChirstmasIs) Define Dictionaries (santasList)	Num allIWantForChirstmas gifts is [1, 2, 3, 4] Num santasList myList is { 'one' 1, 'two' 2, 'three' 3, 'four' 4,}
For (FA LA LA) statement	Num allIWantForChirstmasIs gifts is [1, 2, 3, 4] FALALALALA (value) in gifts Merry Hohoho (value) Christmas
While (‘tis the season of) Statement	Num reindeer reindeer is 10 ‘tis the season of (reindeer <= 10) Merry Hohoho (reindeer) Reindeer += 1 Christmas

2. Lexical structure

2.1 Programs

A Santa ***program*** consists of one or more ***source files***. A source file is an ordered sequence of (probably Unicode) characters.

Conceptually speaking, a Santa program is compiled using three steps

1. Transformation, which converts a file from a particular character repertoire and encoding scheme into a sequence of Unicode characters.
2. Lexical analysis, which translates a stream of Unicode input characters into a stream of tokens.
3. Syntactic analysis, which translates the stream of tokens into executable code.

2.2 Grammars

This specification presents the syntax of the Santa programming language where it differs from Python and Java.

2.2.1 Lexical grammar where different from Python and Java

BNF grammar productions for Santa

The lexical grammar of Santa is very similar to Python with some Java influences. Since Python more heavily influences syntax, the proceeding grammars show the differences between Python and Santa.

<Assignment operator> → is
<Mathematical operator> → + | * | / | -
<Comparison operator> → == | != | <= | >=
<Keyword> → <Language Defined>
→ <Variable Defined>
<Begin Block> → Merry
<End Block> → Christmas

2.2.2 Syntactic ("parse") grammar where different from Python and Java

The syntactic grammar for Santa is similar to a blending of Python and java. The Santa syntactic grammar is outlined below.

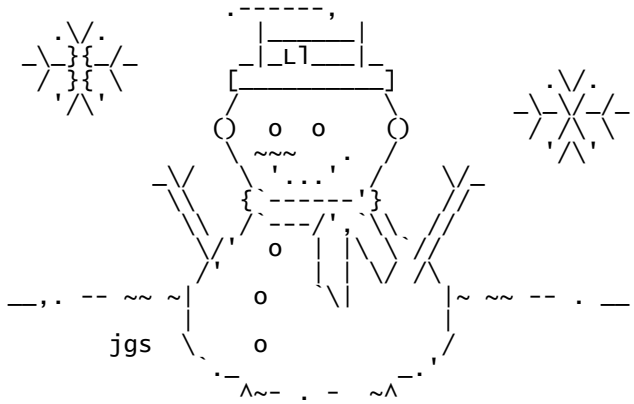
<workshop declaration> → workshop <identifier>
<elf declaration> → <access modifier> elf <identifier>
<method declaration> → <access modifier> <object type> <identifier> <parameter list>
→ <access modifier> <object type> <identifier>
<parameter list> → <parameter> <parameter list>
→ <parameter>
<parameter> → <object type> <identifier>

2.3 Lexical analysis

2.3.1 Comments

There are two forms of comments supported in Santa single-line comments and delimited comments.

Single-line comments start with the characters <<<- (representative of a sideways Christmas tree) and extend to the end of the source line.



Delimited comments were initially going to be an entire keyboard snowman as demonstrated above beside the commented out lines of code. After much consideration, it was decided to change to create delaminated comments to begin with the characters *<<- and end with the characters - >>* (representative of a Christmas tree with a star on top). Delimited comments may span multiple lines. Comments do not nest.

2.4 Tokens

There are several kinds of tokens identifiers, keywords, literals, operators, and punctuators. White space and comments are not tokens, though they act as separators for tokens where needed.

Santa Language Specification

tokens
- identifier
- keyword
- integer-literal
- real-literal
- character-literal
- string-literal
- operator-or-punctuator

2.4.1 Keywords different from Python or Java

A ***keyword*** is an identifier-like sequence of characters that is reserved, and cannot be used as an identifier except when prefaced by the @character.

New Keywords	*New Keywords Continued*	*Removed Keywords:*
• Merry (begin) • Christmas (end) • FALALALALA (for) • ‘tis the season of (while) • elf (class) • Carol (String) • Num (int, double, all number types)	• elfself. (self.) • hohoho (print) • receive (return) • check if (if) • allIWantForChristmas (list / array) • santasList (dict) • is (= assignment) • coal (null)	• for • void • class • int • self • println • true • false • null

3. Types

Santa types are divided into two main categories ***Value types*** and ***Reference types***. (*Maybe you have some other thoughts here. I hope so.)*

3.1 Value types (different from Python and Java)

Num – a general purpose number value type that can be written as an int, fractional, or decimal component

ex Num christmasLights = 10000

Char – a Unicode character, a single component of a Carol

ex. Char firstLetterOfSanta = "S"

Carol – a string which is a set of Chars.
In Santa it is inferred by the programming language that a Carol is an allIWantForChristmas (array) of Chars

ex. Carol toy = "Racecar"

Boolean – a value which can either be **Naughy** (False) or **Nice** (True)

Coal – a null type representing no value

3.2 Reference types (differing from Python and Java)

allIWantForChristmas – a systematic arrangement of data (an array)

ex.
Carol allIWantForChristmas is ["Doll", "Gameboy", "Candy," "Car"]
Num allIWantForChristmas is [1,2,3,4,5]
Combo allIWantForChristmas is [1, "Toy," 5, "Jewelry"]

santasList – an associative array where the keys are of the same type. (*He's checking it twice) aka a dictionary*

ex.
```
Num santasList myList is {
  'one'   1,
  'two'   2,
  'three'   3,
  'four'   4,
}
```

A **santasHelper** is an object which is an instance of an Elf (class). The reference values are pointers to these elves and a special coal reference, which refers to no elf.

4. Example Programs

The Santa programming language in six example programs that demonstrate its use; especially what's new and improved over current languages Java and Python, on which it is based.

4. Examples

4.1 Caesar Cipher Encrypt

```
2   def Carol encrypt(Carol str, Num ShftAmt)
3   Merry
4         gift is ""
5         FALALALALA i in range(len(str))
6         Merry
7               char is str[i]
8               check if (char.isUpper())
9                   gift += chr((ord(char) + ShftAmt -65) % 26 + 65)
10              else
11                  gift += chr((ord(char) + ShftAmt - 97) % 26 + 97)
12              receive gift
13        Christmas
14  Christmas
```

4.2 Caesar Cipher Decrypt

```
1.  def Carol decrypt(Carol Str, Num ShftAmt)
2.  Merry
3.        gift is encrypt(Str, ShftAmt * -1)
4.        receive gift
5.  Christmas
```

4.3 Factorial

```
def Num factorial(Num n)
Merry
    check if n == 0
       received 1
    else
       received n * factorial(n-1)
Christmas
```

4.4 Bubble Sort

```
def allIWantForChirstmas bubbleSort(allIWantForChirstmas gifts)
Merry
    FALALALALA value in range(len(gifts)-1,0,-1)
    Merry
        FALALALALA i in range(value)
        Merry
            check if gifts[i]> gifts [i+1]
                temp is gifts[i]
                gifts [i] is gifts[i+1]
                gifts [i+1] is temp
        Christmas
     Christmas
Christmas


Num allIWantForChirstmas gifts is [54,26,93,17,77,31,44,55,20]
bubbleSort(gifts)
hohoho(gifts)
```

4.5 Quick Sort

```
def partition(gifts,smallest,biggest)
Merry
        i is (smallest-1 )
        pivot is gifts[biggest]
        FALALALALA j in range(smallest, biggest)
        Merry
                check if gifts[j] <= pivot
                        i is i+1
                    gifts[i], gifts[j] is gifts[j], gifts[i]
        gifts[i+1], gifts[biggest] is gifts[biggest], gifts[i+1]
        receive ( i+1 )
        Christmas
Christmas

def quickSort(gifts,smallest,biggest)
Merry
         check if smallest < biggest
               pi is partition(gifts, smallest, biggest)
               quickSort(gifts, smallest, pi-1)
               quickSort(gifts, pi+1, biggest)
Christmas

Num allIWantForChirstmas gifts is [10, 7, 8, 9, 1, 5]
n is len(gifts)
quickSort(gifts,0,n-1)
FALALALALA i in range(n)
```

```
27.         hohoho(gifts[i])
```

4.6 Binary (Christmas) Tree

```
elf Node
Merry
    def Node(elfself, Num val)
    Merry
        elfself.l is Coal
        elfself.r is Coal
        elfself.v is val
    Christmas
Christmas

elf christmasTree
Merry
    def christmasTree(elfself)
    Merry
        elfself.root is Coal
    Christmas

    def getRoot(elfself)
    Merry
        receive elfself.root
    Christmas

    def add(self, val)
    Merry
       check if(elfself.root == Coal)
            elfself.root is Node(val)
        else
            elfself.add(val, elfself.root)
    Christmas

    def add(elfself, val, node)
    Merry
        check if(val < node.v)
            check if(node.l != Coal)
                elfself.add(val, node.l)
            else
                node.l is Node(val)
        else
            check if(node.r != Coal)
                self._add(val, node.r)
            else
                node.r is
 Node(val)
    Christmas
```

```
    def find(self, val)
    Merry
        check if(elfself.root != Coal)
            receive elfself.find(val, elfself.root)
        else
            receive Coal
    Christmas

    def find(self, val, node)
    Merry
        check if(val == node.v)
            receive node
        check else if (val < node.v and node.l != Coal)
            elfself.find(val, node.l)
        check else if(val > node.v and node.r != Coal)
            elfself.find(val, node.r)
    Christmas

    def deleteChristmasTree (elfself)
    Merry
        elfself.root is Coal
    Christmas

    def hohohoChristmasTree(elfself)
    Merry
        check if(elfself.root != Coal)
            elfself.hohohoChristmasTree (elfself.root)
    Christmas

    def hohohoChristmasTree(elfself, node)
    Merry
        check if(node != Coal)
            elfself.hohohoChristmasTree (node.l)
            hohoho Carol(node.v) + ' '
            elfself.hohohoChristmasTree (node.r)
    Christmas
Christmas

christmastree is christmastree()
christmastree.add(3)
christmastree.add(4)
christmastree.add(0)
christmastree.add(8)
christmastree.add(2)
christmastree.hohohoTree()
hohoho (christmastree.find(3)).v
hohoho christmastree.find(10)
christmastree.deleteChristmasTree()
christmastree.hohohoTree()
```

```
*<<- output – A Christmas Binary Tree
                        ^
                <       3       >
        <               0       4               >
  <                     2       8                       >
                        ||
->>*
```