CS421 Project Presentation: Haskython: Haskell-Python Conversion

David Tan Sang Tran
davidtt2@illinois.edu
University of Illinois at Urbana-Champaign

Detailed information can be found on the project report attached or at https://github.com/davidtstran/haskython/blob/main/CS421/421-Project-Report-David-Tran.pdf

Overview of Haskython

- Haskython is a Python library that can convert between Python and Haskell code. https://github.com/davidtstran/haskython

- Motivation:
  - Design an easy way to translate between the two languages.
  - Allow procedural developers to learn functional programming.
  - Utilize the advantages of Haskell through Python code.

- Goals:
  - Create a functioning conversion library for Haskell to Python and Python to Haskell.
  - Provide a public library for ease of use via a pip package.
  - Promote Haskython and encourage other developers to contribute to the project.

Implementation (Tasks and Capabilities)

- Tasks:
    - Take in a file and create an empty file for conversion
    - Parse original file line-by-line
    - Determine function declaration, function parameters, and return type
    - Recognize function body and determine structure
    - Recognize recursion, conditionals, lists, etc.
    - Return the converted line back to the created file

- Detailed code listing can be found in the Code Listing slide.

Implementation (Components)

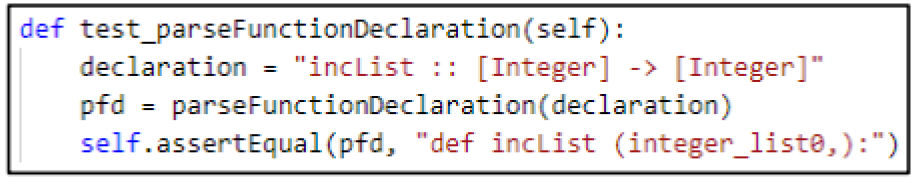
- Components:
  - `haskell_to_python.py`: File used to convert Haskell to Python
  - `python_to_haskell.py`: File used to convert Python to Haskell
  - Files can be read in through the command line call:
    - `python haskell_to_python.py file.hs`
    - `python python_to_haskell.py file.py`

- Detailed code listing can be found in the Code Listing slide.

Implementation (Project Status)

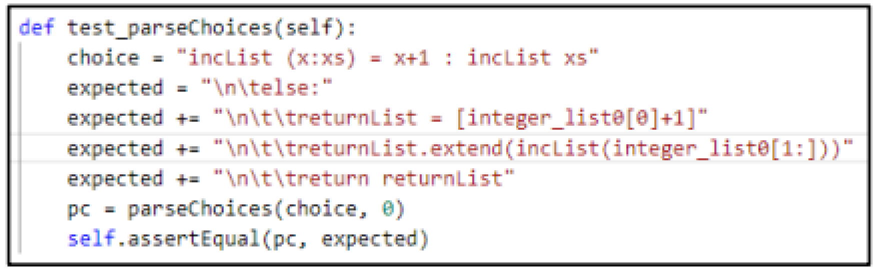
- Works Well:
  - Reading original file line-by-line
  - Creating a new file in the opposing language with the “_haskython” tag
  - Basic Haskell to Python conversion
- Partially Works:
  - Advanced Haskell to Python conversion
- Not Implemented / Future Work:
  - Advanced Python to Haskell conversion
  - Create pip package
  - Collaborate with other developers as an open-source project

Tests (Unit Tests)



```
def test_parseFunctionDeclaration(self):
    declaration = "incList :: [Integer] -> [Integer]"
    pfd = parseFunctionDeclaration(declaration)
    self.assertEqual(pfd, "def incList (integer_list0,):")
```

- Test Parse Function Declaration
  - In the above code snippet, this test will retrieve the function declaration from the Haskell file and convert it into a Python function declaration using the `def` keyword and a counter to distinguish various parameters. The results can be seen in the function `assertEqual()`. With this test passing, it is proven that the library will be able to successfully convert a Haskell functional declaration into Python.

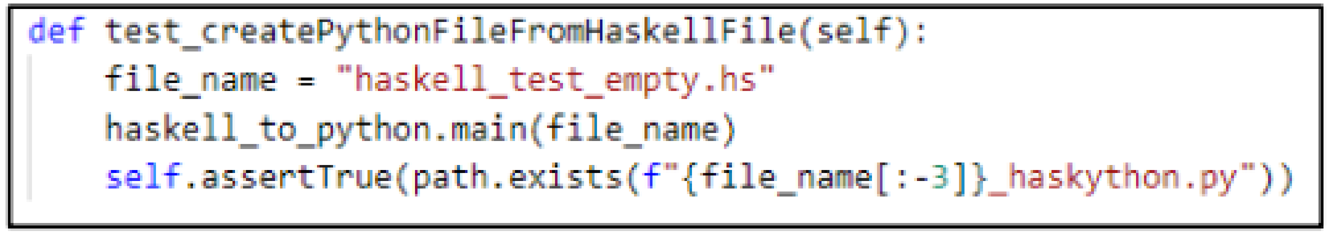


```
def test_parseChoices(self):
    choice = "incList (x:xs) = x+1 : incList xs"
    expected = "\n\telse:"
    expected += "\n\t\treturnList = [integer_list0[0]+1]"
    expected += "\n\t\treturnList.extend(incList(integer_list0[1:]))"
    expected += "\n\t\treturn returnList"
    pc = parseChoices(choice, 0)
    self.assertEqual(pc, expected)
```

- Test Parse Choices
  - In the above image, this test will retrieve a function body from the Haskell file and convert it into Python. For this test, we will use the `incList (x:xs)` body to create the Python code that will create a list with the first element incremented with recursion to create the same output as the Haskell `incList` function. This test shows that after a function declaration, the Python script can convert a function body from Haskell to Python.

Tests (Feature Tests)

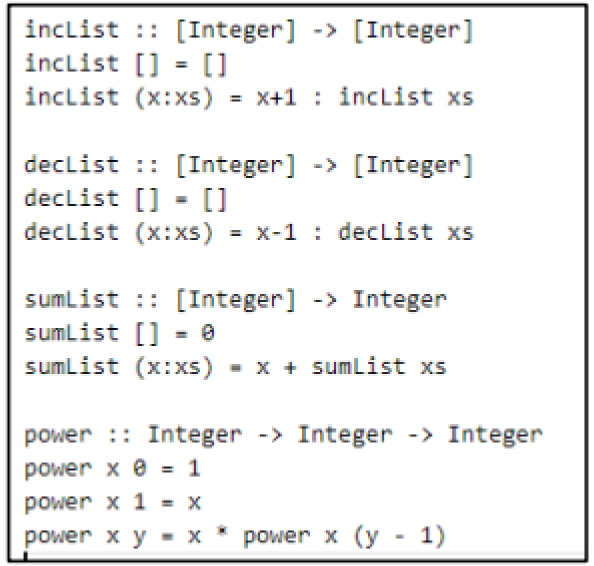
- Test Creating New File with _Haskython Tag
    - With the test, it shows that the project is able to create new file from `haskell_test_empty.hs` in the same directory with the correct extension of `_haskython.py`. The output file would be called `haskell_test_empty_haskython.py`. With the file being correctly created, we can ensure that the Haskell file is converted into a Python file in the same directory where the file is added. The next subsection will test the functionality of the created Python file.



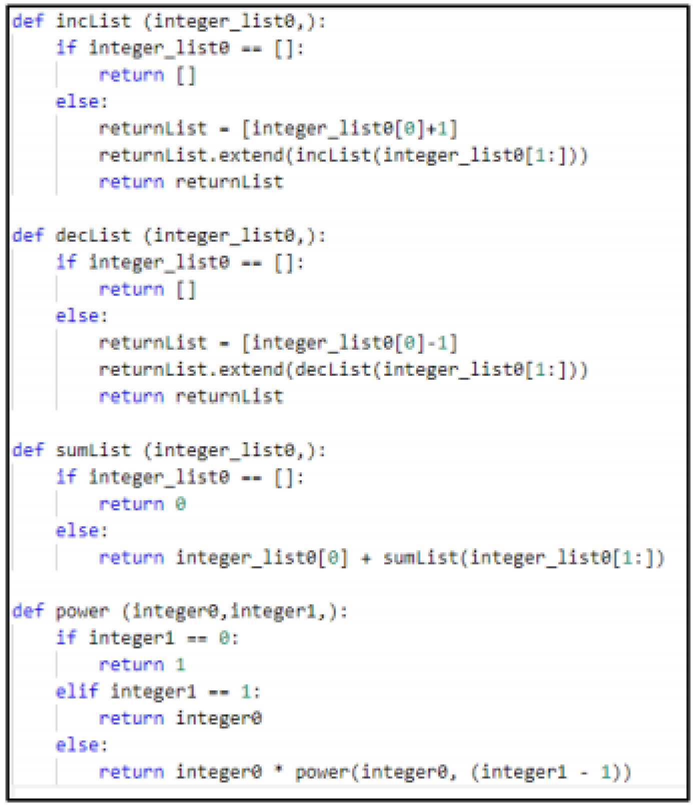
```python
def test_createPythonFileFromHaskellFile(self):
    file_name = "haskell_test_empty.hs"
    haskell_to_python.main(file_name)
    self.assertTrue(path.exists(f"{file_name[:-3]}_haskython.py"))
```

Tests (Larger Tests) – Part 1

- First, Convert the `haskell_test.hs` to `haskell_test_haskython.py` using `haskell_to_python.py`



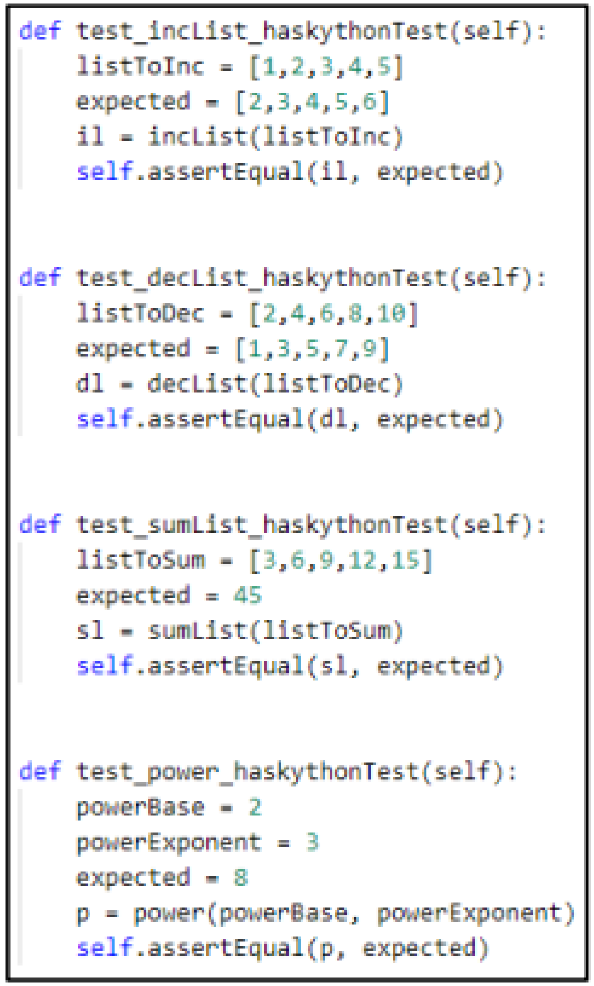
```haskell
incList :: [Integer] -> [Integer]
incList [] = []
incList (x:xs) = x+1 : incList xs

decList :: [Integer] -> [Integer]
decList [] = []
decList (x:xs) = x-1 : decList xs

sumList :: [Integer] -> Integer
sumList [] = 0
sumList (x:xs) = x + sumList xs

power :: Integer -> Integer -> Integer
power x 0 = 1
power x 1 = x
power x y = x * power x (y - 1)
```



```python
def incList (integer_list0,):
    if integer_list0 == []:
        return []
    else:
        returnList = [integer_list0[0]+1]
        returnList.extend(incList(integer_list0[1:]))
        return returnList

def decList (integer_list0,):
    if integer_list0 == []:
        return []
    else:
        returnList = [integer_list0[0]-1]
        returnList.extend(decList(integer_list0[1:]))
        return returnList

def sumList (integer_list0,):
    if integer_list0 == []:
        return 0
    else:
        return integer_list0[0] + sumList(integer_list0[1:])

def power (integer0,integer1,):
    if integer1 == 0:
        return 1
    elif integer1 == 1:
        return integer0
    else:
        return integer0 * power(integer0, (integer1 - 1))
```

Tests (Larger Tests) – Part 2

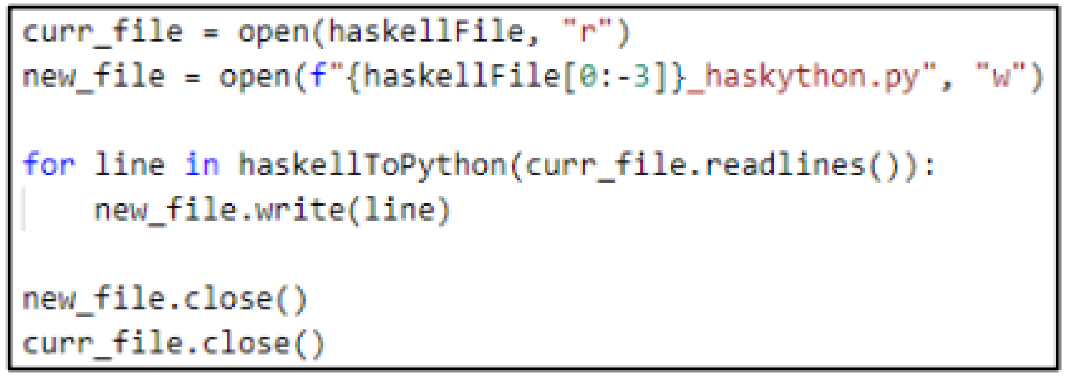
- With all of these tests passing as expected, it can be determined that the Haskell to Python is working as expected including reading a Haskell file, collecting the file information, generating the new Python file, and validating that the new functions perform successfully.



```
def test_incList_haskythonTest(self):
    listToInc = [1,2,3,4,5]
    expected = [2,3,4,5,6]
    il = incList(listToInc)
    self.assertEqual(il, expected)


def test_decList_haskythonTest(self):
    listToDec = [2,4,6,8,10]
    expected = [1,3,5,7,9]
    dl = decList(listToDec)
    self.assertEqual(dl, expected)


def test_sumList_haskythonTest(self):
    listToSum = [3,6,9,12,15]
    expected = 45
    sl = sumList(listToSum)
    self.assertEqual(sl, expected)


def test_power_haskythonTest(self):
    powerBase = 2
    powerExponent = 3
    expected = 8
    p = power(powerBase, powerExponent)
    self.assertEqual(p, expected)
```

Code Listing (Central Function 1): Read Haskell File, Write to Python Conversion File

- The following code is a core function that reads in a Haskell file and create a new Python file to add the translated lines.

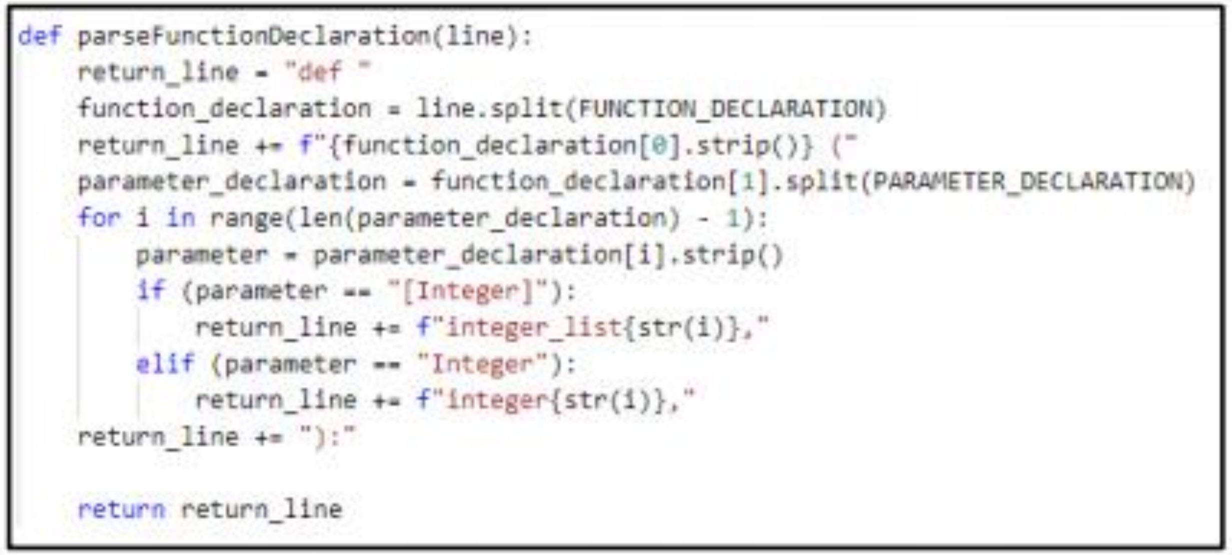


```
curr_file = open(haskellFile, "r")
new_file = open(f"{haskellFile[0:-3]}_haskython.py", "w")

for line in haskellToPython(curr_file.readlines()):
    new_file.write(line)

new_file.close()
curr_file.close()
```

- Detailed information can be found on the project report attached or at https://github.com/davidtstran/haskython/blob/main/CS421/421-Project-Report-David-Tran.pdf

Code Listing (Central Function 2): Recognize Function Declarations, Parameters, and Types

- The following code is a core function that recognizes a line in the Haskell file as a function definition and then parses the parameters and return type.



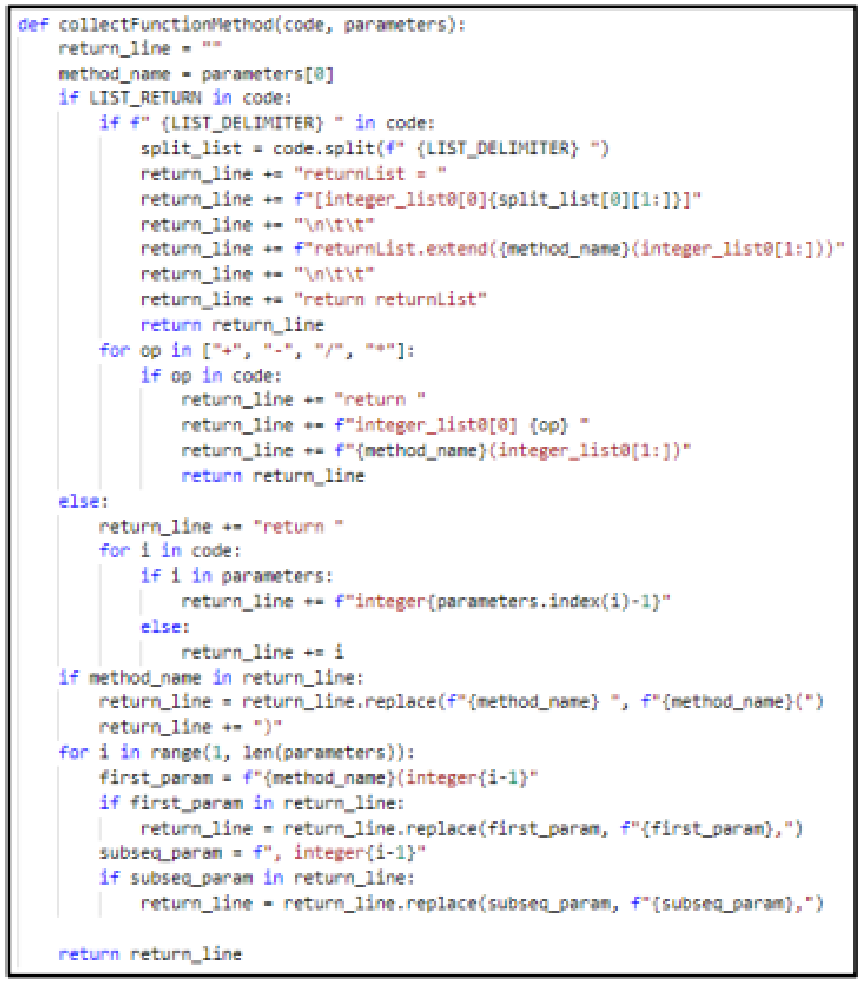
```python
def parseFunctionDeclaration(line):
    return_line = "def "
    function_declaration = line.split(FUNCTION_DECLARATION)
    return_line += f"{function_declaration[0].strip()} ("
    parameter_declaration = function_declaration[1].split(PARAMETER_DECLARATION)
    for i in range(len(parameter_declaration) - 1):
        parameter = parameter_declaration[i].strip()
        if (parameter == "[Integer]"):
            return_line += f"integer_list{str(i)},"
        elif (parameter == "Integer"):
            return_line += f"integer{str(i)},"
    return_line += "):"

    return return_line
```

- Detailed information can be found on the project report attached or at https://github.com/davidtstran/haskython/blob/main/CS421/421-Project-Report-David-Tran.pdf

Code Listing (Central Function 3): Recognize Conditionals, Loops, Lists, and Recursion

- The following code is a core function that performs inside a function body to recognize traits such as conditionals, loops, lists, recursions, etc.

- Detailed information can be found on the project report attached or at https://github.com/davidtstran/haskython/blob/main/CS421/421-Project-Report-David-Tran.pdf



```python
def collectFunctionMethod(code, parameters):
    return_line = ""
    method_name = parameters[0]
    if LIST_RETURN in code:
        if f" {LIST_DELIMITER} " in code:
            split_list = code.split(f" {LIST_DELIMITER} ")
            return_line += "returnList = "
            return_line += f"[integer_list0[0]{split_list[0][1:]}]"
            return_line += "\n\t\t"
            return_line += f"returnList.extend({method_name}(integer_list0[1:]))"
            return_line += "\n\t\t"
            return_line += "return returnList"
            return return_line
        for op in ["+", "-", "/", "*"]:
            if op in code:
                return_line += "return "
                return_line += f"integer_list0[0] {op} "
                return_line += f"{method_name}(integer_list0[1:])"
                return return_line
    else:
        return_line += "return "
        for i in code:
            if i in parameters:
                return_line += f"integer{parameters.index(i)-1}"
            else:
                return_line += i
    if method_name in return_line:
        return_line = return_line.replace(f"{method_name} ", f"{method_name}(")
        return_line += ")"
    for i in range(1, len(parameters)):
        first_param = f"{method_name}(integer{i-1}"
        if first_param in return_line:
            return_line = return_line.replace(first_param, f"{first_param},")
        subseq_param = f", integer{i-1}"
        if subseq_param in return_line:
            return_line = return_line.replace(subseq_param, f"{subseq_param},")

    return return_line
```

Conclusion

- Detailed information can be found on the project report attached or at https://github.com/davidtstran/haskython/blob/main/CS421/421-Project-Report-David-Tran.pdf

- The Haskython project provides an easy and quick way to convert back and forth between Haskell and Python.
- The functions cover the majority of basic Haskell functionalities and will be continuously added upon to provide a deeper level of usage.
- Haskython is created with the thought that procedural programmers can have an opportunity to slowly learn functional programming through the translation process

References

- Tran, David T.S., “Haskython: Haskell and Python Conversion (Python Library)”. GitHub, Inc. CS421: Programming Languages Course. Summer 2021. https://github.com/davidtstran/haskython
- Beckman, Mattox, “CS 421: Programming Languages” Coursera. University of Illinois at Urbana-Champaign. Summer 2021. https://www.coursera.org/learn/cs421/home/welcome
- Python Wiki, “Python vs Haskell”. MoinMoin Wiki Engine. Python. https://wiki.python.org/moin/PythonVsHaskell
- D. Can, V. R. Martinez, P. Papadopoulos and S. S. Narayanan, "Pykaldi: A Python Wrapper for Kaldi," 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018, pp. 5889-5893, doi: 10.1109/ICASSP.2018.8462463. https://ieeexplore.ieee.org/document/8462463
- S. Behnel, R. Bradshaw, D. S. Seljebotn, G. Ewing, W. Stein, G. Gellner, et al. “Welcome to Cython’s Documentation”. Cython 3.0.0a9 Documentation. https://cython.readthedocs.io/en/latest/