# auxi
# Lending a Helping Hand to Improve Metallurgical Processes

J.H. Zietsman[1,2] and C. Kok[1]

[1]Ex Mente (Pty) Ltd
[2]Department of Materials Science and Metallurgical Engineering, University of Pretoria

**Abstract**

Process calculations are critical to the successful monitoring and operation of metallurgical plants, and an important part of a metallurgical process engineer's routine duties. These calculations can range from simple stoichiometry, heat transfer, recovery, and yield calculations, through mass and energy balances, to sophisticated models of process units and circuits. Historically, metallurgical process calculations have been done on paper, on the "back of a cigarette box", and in recent times with spreadsheets.

The mining and metallurgical industry is of paramount importance to South Africa. At this point, due to various reasons, our industry and our economy are in a dire state. It is with this background that we started developing a freely available, open-source toolbox of easy-to-use, powerful calculations to help metallurgical process engineers to perform their work more productively, accurately and effectively to the benefit of their employers, and ultimately our economy.

`auxi` is implemented and used in Python, a simple but powerful open-source programming language. Python was selected because it is simple enough to be used practically in a metallurgical plant environment. It also has a wealth of on-line training material, and libraries for scientific and engineering computing. This covers areas such as linear algebra, data analysis, plotting, numerical methods, and now through `auxi`, metallurgical process calculations.

The `auxi` initiative is about more than calculations, though. The tools provided are simple, but important. They are worth having, but perhaps not worth paying for... at least not in the traditional sense. The open-source software community has demonstrated the significant impact that collaborating communities can have on the advancement of technology. We hope that `auxi` will, in some small way, help advance our metallurgical industry through sharing, collaboration and innovation.

## Introduction

South Africa is endowed with an abundance of mineral resources. The mining, minerals and metals (MMM) industry has been contributing significantly to the country's economy for many years, and we are still strongly dependent on its success. In 2015, mining contributed 8.7 % to the national GDP, and manufacturing 14.1 %. This places the total impact of mining, minerals and metals between 8.7 and 22.8 % (Statistics South Africa 2016b). At the beginning of 2016, this industry was responsible for 42.5 % of the country's exports. (Statistics South Africa 2016a).

This sector of the South African economy has been plagued by serious difficulties in recent years. Labour unrest, electrical energy supply problems, and international market conditions have all contributed to reduced productivity and competitiveness, and ultimately plant closures and job losses. Solutions to these problems are not in mineral riches and high commodity prices. We need to seek for it in our people, through creativity and innovation. People are the real riches of any country, and ultimately determine the strength of an economy. As both the current and previous South African mottos suggest (*!ke e: /xarra //ke* and *ex unitate vires*), we believe the only way to face these challenges is to unite, share and work together.

In this article we present an open-source software toolkit for doing metallurgical process calculations. It is a small contribution to solving problems and raising the standard in our industry. We hope that it can be a spark that can start a much needed fire.

## The `auxi` Initiative

The software toolkit presented here is named `auxi`. This is derived from the Latin words for help (*auxilio*) and helper (*auxilium*). The intention with the initiative is to provide help to students, engineers, technicians and operations personnel involved in the South African metallurgical industry, whether it be in education, research, engineering or production environments.

`auxi` can assist metallurgical process engineers since there are not many tools that they can use in day to day tasks such as calculations, analysing processes and improving operations. There are software products available, such as METSIM, Pyrosim, IDEAS, HSC, FactSage, and many others. In many cases these tools are not available and too expensive to buy.

Most process engineering graduates entering industry arrive with knowledge and some skills they obtained during their studies, but usually with no tools for the job. Their tool of choice is most often spreadsheets, which contains little or no tried and tested functions or algorithms specifically suited to metallurgical process engineering.

`auxi` is written and used in the Python programming language (Python Software Foundation 2016). It very different from using a spreadsheet, but it also has definite advantages. Tools can be easily packaged and distributed, and the functions and algorithms can be tested to ensure that they contain no errors. It is no longer necessary to type in molar masses and enthalpy equations every time. They are available when you start doing a new task or project.

Very importantly, Python is open source and therefore freely available to download and use. The University of Pretoria teaches all engineering graduates Python programming in their second year. When they arrive in industry, they are therefore able to start working with the tools they learned during their training without having to motivate the purchase of expensive software packages. They can become productive and make contributions quicker.

By building basic metallurgical process calculation tools into `auxi`, we hope that process engineers will be more confident and effective in doing these calculations. We need them to spend less time focusing calculation details in, for example, mass and energy balances, and more time solving challenging industrial problems and innovation.

By making `auxi` open source, we hope to inspire the South African metallurgical process engineering community to share, work together and improve our industry. Companies would often not be willing to pay for such tools. If we build them together and share, it would be a small price to pay, and the entire community can benefit.

## Python

Python in is an open-source programming language conceived in the late 1980s. Its first version was released in 1994, and it currently has two actively used versions, namely 2.7 and 3.x (Wikipedia 2016). We decided to implement `auxi` in Python for the following reasons:

**Accessibility:** Python is open source and freely available, which makes it possible to give anyone with access to a computer access to Python. Commercial software packages make it more difficult for a community to collaboratively develop and freely share resources.

**Availability:** Python runs on most computer platforms. This includes Windows, OS X, and Linux. Code written on one platform can be easily transferred to another. It even runs on Apple and Android smartphones.

**Simplicity:** In the simplest case, you can use Python interactively, almost like a calculator. Its syntax is clean and compact, which means you can do more and type less. It is well-suited for writing pieces of code quickly to do small things, like automate tedious data analysis tasks and perform basic calculations.

**Rapid development:** Python is an interpreted language, which means a program can be run directly without having to build an executable file first and running that. This, and the language's compact syntax makes it possible to write programs much quicker than in most other languages.

**Power:** Even though Python is simple and compact, the language is also powerful enough to do big things, like writing extensive libraries and complex applications. Its object orientated capabilities are an important feature that makes this possible.

**Versatility:** Our work environments tend to grow in complexity. We may need to interact with the internet, large data sets from laboratories or information systems, and even laboratory or plant equipment. Python is able to do all of these things and more in a single environment. It is the perfect "glue language".

**Reliability:** Spreadsheet calculations are hard to test and verify. It is easy to mistakes and hard to find them. This could have dire consequences if calculations are used to make decisions on a plant or large engineering project. Python has powerful test frameworks, which make it possible to formally test and verify calculations. This improves reliability and accuracy.

**Support:** Because Python has been in use for more than 20 years, you are never alone when experiencing problems. There is extensive online documentation available, as well as large international community that exchange ideas and solve problems on forums like stackoverflow.com.

**Available packages:** The most compelling reason for using Python is the large number of packages that are freely available. These packages provide tools for mathematics, data analysis, data visualisation, and virtually anything one can think of. It is in this spirit that `auxi` contributes metallurgical calculations so that ever more complex problems can be solved quickly and effectively.

## `auxi` Overview

So, what is `auxi`? It is a Python package (a toolkit or toolbox) that you can download and install on your computer. It contains functions and classes (tools) that you can use to perform a wide range of metallurgical process calculations.

In South Africa, metallurgical calculations are usually done in Excel spreadsheets. Using `auxi` therefore requires seasoned Excel users to start using Python. This is likely to be challenging for many, and providing a powerful toolbox with open source code is simply not enough. For this reason `auxi` consists of the following components:

**Python package:** The package is hosted on PyPI, the Python Package Index, at `https://pypi.python.org/pypi/auxi`. You can download and install it on your computer with one simple command, and be up and running in minutes.

The `auxi` package contains all the source code as well as embedded documentation. You don't need to view the source code, but if you want to understand how the calculations work, you can. It is not a black box.

**Source code:** The source code is hosted and maintained on Github, at `https://github.com/Ex-Mente/auxi.0`. It is managed with the git source control system to keep a record of all revisions, and to make it possible for any member of the community to contribute.

The `auxi` code repository is moderated by administrators. They ensure that all code that is added to auxi, is properly formatted, tested and documented.

**Documentation:** This is a critical part of any system. A comprehensive set of documentation is available at `http://auxi.readthedocs.org/en/latest/`. You can read it with a web browser, or download a PDF document that can be printed.

**Training:** We have all heard the phrase: "If all else fails, read the manual." The reality is that many people don't like reading manuals, neither for washing machines, nor for software.

For those of us who prefer being shown how things work rather than read through documentation, there is an `auxi` YouTube channel at `https://www.youtube.com/channel/UCdklSCJ8S9wFyayLAO7iINQ`. It contains short videos of around 5 minutes each, demonstrating how to get started, how to add data, and how to perform specific types of calculations.

**Support:** When using virtually any tool, you are bound to find yourself in a situation where neither the documentation nor the training is sufficient to solve your problem. This is when we need someone else's help, someone with more experience.

The `auxi` Q&A forum is a Google group where users can ask questions, and other users and developers can answer them. It is available at `https://groups.google.com/forum/#!forum/auxi-za`. Because `auxi` is not commercial software, it depends on the community to make it work.

**License:** Some open source software allows non-commercial use only. Users invest time to learning it, only to find out that their hands become tied when they want to use it for commercial ends. This can be frustrating and disappointing.

`auxi`'s development, distribution and use is governed by the GNU Lesser General Public License, version 3 (Free Software Foundation 2007). This allows both commercial and non-commercial use, and protects the investment a user makes, both in terms of time and additional software that is developed.

**Community:** The above components are all necessary, valuable and important. None of them is, however, as important as the community of developers and users. A healthy, actively involved community will produce a valuable set of tools that continues to grow. This is what causes an initiative like `auxi` to either flourish or die.

Collaborative projects like `auxi` present the metallurgical engineering community in South Africa with an opportunity to work together on matters of common interest, which will ultimately benefit the country. It is a mind-shift that can improve our competitiveness.

# `auxi` Content

When developing `auxi`, we address the most simple things first. Once these are in place, we use them to create more powerful tools that can perform more complex tasks in simple ways. Figure 1 shows an overview of the content of `auxi` version 0.2.0. This structure will grow in future versions.

In Python, packages and sub-packages are equivalent to folders, and modules are equivalent to files. The functions, subroutines and classes that do the actual work, reside in modules.

## Tools

The tools package contains components to do basic tasks. Version 0.2.0 only contains a single sub-package, namely chemistry.

The chemistry package provides stoichiometry and thermochemistry tools that are often used in simple calculations, but also in mass and energy balances, and in sophisticated process models. The functions available in the stoichiometry module are listed in Table 1, and those in the thermochemistry module in Table 2.

Version 0.2.0 only contains thermochemical data from Rao (1985) and Dinsdale (1991). This will be expanded in future. It is also possible to import pure substance data from FactSage (Bale et al. 2009).

## Process Modelling

The process modelling package provides classes to do calculations with different types of materials. It currently contains the following modules:

**psd:** This module is used to describe particulate materials using only a particle size distribution. It is useful in comminution and dry separation circuits.

**psdslurry:** In mineral processing operations, material is often handled as a slurry. This module describes materials that consist of a combination of water and particulate solids. The solid portion is described with a particle size distribution.

**chem:** The `chem` module describes materials with a chemical composition. This is useful when calculating mass balances of chemically reactive systems when an energy balance is not required.

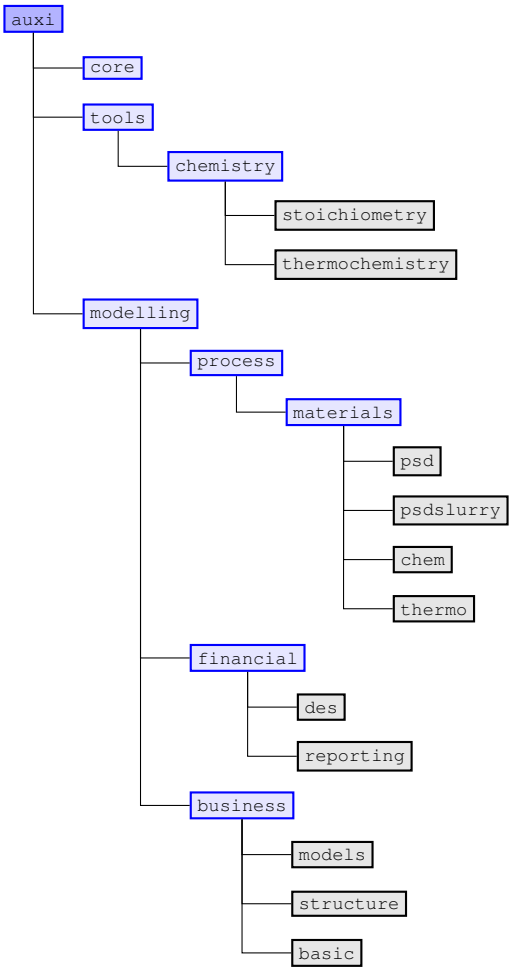


Figure 1: An overview of the `auxi` 0.2.0 package (dark blue), its sub-packages (light blue) and modules (grey).

**thermo:** This module is useful for pyrometallurgical processes that require accounting of both mass and energy. Enthalpy calculations are integrated in the classes, and done automatically.

Each of the above modules contain two classes, namely `Material` and `MaterialPackage`. A `Material` describes the structure of a material. This would be a list of size classes for particulate materials, or a list of chemical compounds for chemical materials. A `Material` can contain assays that describe, for example, iron ores from different mines, or size distributions from different parts of a circuit.

A `MaterialPackage` represents a specific quantity of material, such as 1000 kg of Sishen iron ore of a specific composition, or 10 kg of the undersize of a specific screen in a comminution circuit.

A `Material` object is used to create `MaterialPackages`, avoiding the need for configuring numerous calculations to multiply the required quantity with a composition and calculating enthalpies. The `MaterialPackages` are then manipulated by splitting and combining them. Rather than configuring calculations focused on individual compounds, this makes it possible to focus on how the material is manipulated by the process in question.

This approach achieves two benefits. Firstly, it re-uses well-tested calculations in the classes that prevent potential calculation errors. Secondly, it focuses the user more on the process and how it functions, and less on basic calculations. `auxi`'s process modelling package therefore takes care of many of the mundane calculations that you would have to configure by hand in a spreadsheet. This saves time and improves accuracy.

Table 1: Functions available in the stoichiometry tools module.

| Function | Parameters | Description and Example | Calculation |
|---|---|---|---|
| amount | compound<br>mass | Calculate the amount (kmol) of the specified mass (kg) of a compound $c$. | $n_c = \frac{m_c}{M_c}$ |
| convert_compound | mass<br>source<br>target<br>element | Calculate the mass ($m_t$) of a target compound by converting the specified mass ($m_s$) of a source compound, using element $e$ as basis. | $m_t = m_s \frac{y_{s,e}}{y_{t,e}}$ |
| element_mass_fraction | compound<br>element | Calculate the mass fraction of element $e$ in compound $c$. | $y_{c,e} = \frac{n_{c,e} M_e}{M_c}$ |
| element_mass_fractions | compound<br>elements | Calculate the mass fraction of each element in the elements list in compound $c$. | |
| elements | compounds | Calculate a list of elements that are present in the specified list of compounds. | |
| mass | compound<br>amount | Calculate the mass (kg) of the specified amount (kmol) of compound $c$. | $m_c = n_c M_c$ |
| molar_mass | compound | Calculate the molar mass of compound $c$, given its chemical formula. | $M_c = \sum n_{c,e} M_e$ |
| stoichiometry_coefficient | compound<br>element | Calculate the stoichiometry coefficient of element $e$ in compound $c$, given its chemical formula. | |
| stoichiometry_coefficients | compound<br>elements | Calculate the stoichiometry coefficients of the list of elements in compound $c$, given its chemical formula. | |

Table 2: Functions available in the thermochemistry tools module.

| Function | Parameters | Description and Example | Calculation |
|---|---|---|---|
| Cp | compound<br>temperature<br>mass | Calculate the heat capacity (kWh/K) of the specified mass (kg) of a compound $c$ at temperature $T$. | $c_{p,c}(T) = a + bT + cT^2 ...$ |
| H | compound<br>temperature<br>mass | Calculate the enthalpy (kWh) of the specified mass (kg) of a compound $c$ at temperature $T$. | $H_c(T) = \int_{T_{ref}}^{T} c_p(T) dT$ |
| S | compound<br>temperature<br>mass | Calculate the enthalpy (kWh) of the specified mass (kg) of a compound $c$ at temperature $T$. | $S_c(T) = \int_{T_{ref}}^{T} \frac{c_p(T)}{T} dT$ |
| G | compound<br>temperature<br>mass | Calculate the Gibbs free energy (kWh) of the specified mass (kg) of a compound $c$ at temperature $T$. | $G_c(T) = H_c(T) - TS_c(T)$ |

## Financial Modelling

The financial modelling package contains two modules. The `des` (double-entry system) module provides a double-entry accounting system. The core of this system is a `GeneralLedger` that contains a set of `Accounts`. `Transactions` are created and posted to the `GeneralLedger`.

The `reporting` module contains report generators that produce reports based on the transactions in the `GeneralLedger`. These currently include the following:

**`GeneralLedgerStructure`** displays the set of accounts in the `GeneralLedger` in a table to provide an overview.

**`TransactionList`** lists the details of all the `Transactions` in the `GeneralLedger` between specified start and end dates. This is useful when trying to trace whether a model is functioning correctly, or when trying to explain issues identified in the balance sheet or income statement.

**`BalanceSheet`** displays the status of the balance sheet at a specified end date. This is useful when considering the change in assets, liabilities and equity in a business.

**`IncomeStatement`** displays the income, expenses and profit/loss between the specified start and end dates. This answers the bottom line question about whether the business is successful.

### Business Modelling

The business modelling package supports the creation of business models, which can be either simple or complex. The `models` module currently contains only a single type, namely `TimeBasedModel`. It is used to create discounted cash flow (DCF) models.

The `structure` module contain three classes, namely `Entity`, `Component`, and `Activity`. These classes are used to build the structure of a business model.

An `Entity` represents a business entity, such as a company, closed corporation, partnership, or even a natural person. A `TimeBasedModel` can contain any number of `Entities`. This makes it possible to build models of complex corporations that own subsidiaries, while keeping each company separate. This modular approach limits complexity.

Each `Entity` has its own `GeneralLedger`. This ensures that the financial performance of each `Entity` can be tracked and analysed separately.

An `Entity` is divided into a number of `Components`. These components can represent different areas, such as a mine, concentrator, smelter and refinery. It can also be used to represent different functions or departments, such as human resources, finance, marketing, etc. `Components` can be subdivided into more `Components`, with no limit to the depth of the hierarchy. This again adds modularity to a model, limiting its complexity.

Financial reports can be generated on a specific `Component` to allow zooming into the detail of an area or department in a business. This makes detailed analysis possible without making any changes to a model.

The `Components` in a business contain `Activities`. Examples of business activities include purchases, sales, loans, salary payments and any other action that results in transactions being recorded in the `GeneralLedger`. `Activities` are what makes it faster and more reliable to build business models in `auxi`.

When a business takes out a loan, for example, a `BasicLoanActivity` is configured by specifying its bank account, loan account and interest account in the `GeneralLedger`, and then specifying the loan amount, interest rate, start date and duration. This adds up to seven pieces of information. The `BasicLoanActivity` will then generate all the necessary transactions in the `GeneralLedger` without further actions from the user.

It is possible to create user-defined activities. This provides limitless flexibility. It is possible, for example, to create an activity that runs a process model to determine how much consumables are used and how much product is produced in a specific month or year. These custom activities are used to integrate technical models into business models.

This approach makes activities simple to configure, while reducing the likelihood of making mistakes such as those caused by repeating formulas in spreadsheet cells. The end result is rapid model development and more accurate results.

When a `TimeBasedModel` is run, it executes all the business `Entities` inside it. In turn, each `Entity` executes all of its business `Components`, and each `Component` executes all the activities it contains. The end result is that transactions are posted to the `Entities`' `GeneralLedgers`. The financial reports are then used to analyse the results without having to write additional code.

## `auxi` Examples

In this section examples are presented to demonstrate of how `auxi` is used to work with `Materials` and `MaterialPackages`, and how to build a simple business model. Stoichiometry and thermochemistry calculations are not demonstrated, since these are simply done by calling the functions listed in Table 1 and Table 2.

### Working with Materials and Material Packages

The classes in the `thermo` module are used in this example. Two ilmenite materials are used. Their details are provided in Listing 1.

A new material called `ilmenite` is created from the text file in line 8 of Listing 2. This new material is used to create a package of ilmenite based on the `IlmeniteA` assay, a mass of 300 kg, a pressure of 1 atm and a temperature of 25 °C (line 11). A second package is created based on the `IlmeniteB` assay,

with a mass of 500 kg and temperature of 500 °C (line 12). The two packages are added together to produce a new package in line 14, and the content of this package is printed.

Listing 3 shows the content of the final `MaterialPackage`. In only a few lines of code, two material packages of different compositions, masses and temperatures were created, added together, and the resulting state calculated. This included enthalpy calculations, and determining compound masses, mass fractions and mole fractions. The code focuses on materials and material packages, which relate more closely to the process than do the detailed calculations. This, equips process engineers to develop calculations and models quickly and reliably.

Listing 1: The content of a text file specifying the chemical compounds and assays of an ilmenite material. The name of the file is "ilmenite.txt". Assays are expressed in mass fractions.

```
Compound    IlmeniteA   IlmeniteB
Fe2O3[S1]   0.23000     0.48000
Fe3O4[S1]   0.00000     0.00000
FeO[S]      0.28000     0.19000
SiO2[S1]    0.01000     0.04000
TiO2[S1]    0.48000     0.29000
```

Listing 2: A script that demonstrates how the `Material` and `MaterialPackage` classes in the `thermo` module are used.

```
1  from auxi.tools.chemistry import thermochemistry as th
2  from auxi.modelling.process.materials.thermo import Material, MaterialPackage
3
4  # Load thermochemical data exported from FactSage.
5  th.load_data_factsage("./data")
6
7  # Create a material from a text file.
8  ilmenite = Material("Ilmenite", "./materials/ilmenite.txt")
9
10 # Create two material packages
11 mp1 = ilmenite.create_package("IlmeniteA", mass=300.0, P=1.0, T=25.0)
12 mp2 = ilmenite.create_package("IlmeniteB", mass=500.0, P=1.0, T=500.0)
13
14 mptot = mp1 + mp2
15 print(mptot)
```

Listing 3: Results produced by the code in Listing 2.

```
MaterialPackage

Material            Ilmenite
Mass                 8.00000000e+02 kg
Amount               8.42788204e+00 kmol
Pressure             1.00000000e+00 atm
Temperature          3.30640109e+02  C
Enthalpy            -1.61325683e+03 kWh

Compound Details
Formula             Mass              Mass Fraction     Mole Fraction
Fe2O3[S1]           3.09000000e+02    3.86250000e-01    2.29597527e-01
Fe3O4[S1]           0.00000000e+00    0.00000000e+00    0.00000000e+00
FeO[S]              1.79000000e+02    2.23750000e-01    2.95625342e-01
SiO2[S1]            2.30000000e+01    2.87500000e-02    4.54201309e-02
TiO2[S1]            2.89000000e+02    3.61250000e-01    4.29357000e-01
```

## Business Model

Listing 4 demonstrates how a simple model of a courier business is configured and executed. In lines 10 to 17, the general ledger structure is configured. In lines 20 to 28 the model and business structure are created. The business activities are created in lines 31 to 57. Finally, the model is run and two reports are generated. The reports are shown in Table 3 and Table 4. The reports can be created in different formats, including text, CSV and LaTeX.

Listing 4: A simple model of a courier business.

```
from datetime import datetime
from dateutil.relativedelta import relativedelta

from auxi.core.time import TimePeriod
from auxi.modelling.business.models import TimeBasedModel
from auxi.modelling.business.basic import BasicActivity, BasicLoanActivity
from auxi.modelling.financial.des import GeneralLedgerStructure

# Create general ledger structure and accounts.
gl_structure = GeneralLedgerStructure("Courier GL Structure")

gl_structure["Long Term Borrowing"].create_account("Capital Loan", "0000")
gl_structure["Expense"].create_account("Interest Expense", "0000")
gl_structure["Fixed Assets"].create_account("Vehicle Asset", "0000")
gl_structure["Sales"].create_account("Sales Delivery", "0000")
gl_structure["Cost of Sales"].create_account("Fuel", "0000")
gl_structure["Expense"].create_account("Wages", "0000")

# Create the business model, entity and components.
start_datetime = datetime(2016, 2, 1)
end_datetime = datetime(2021, 1, 1)

model = TimeBasedModel("Business Model", start_datetime=start_datetime,
                       period_duration=TimePeriod.month, period_count=61)

courier_company = model.create_entity("CourierZA", gl_structure=gl_structure)
ops = courier_company.create_component("Operations")
hr = courier_company.create_component("HR")

# Create activities
loan = BasicLoanActivity("Capital Loan",
    bank_account="Bank/Default", loan_account="Long Term Borrowing/Capital Loan",
    interest_account="Expense/Interest Expense",
    amount=200000, interest_rate=0.15, start=start_datetime, duration=36,
    interval=1)
ops.add_activity(loan)

purchase_vehicle = BasicActivity("Purchase Vehicle",
    dt_account="Fixed Assets/Vehicle Asset", cr_account="Bank/Default",
    amount=177000, start=start_datetime, end=start_datetime + relativedelta(months=1),
    interval=1)
ops.add_activity(purchase_vehicle)

make_delivery_sale = BasicActivity("Make Delivery",
    dt_account="Bank/Default", cr_account="Sales/Sales Delivery",
    amount=5000, start=start_datetime, end=end_datetime, interval=1)
ops.add_activity(make_delivery_sale)

pay_delivery_costs = BasicActivity("Pay for Fuel",
    dt_account="Cost of Sales/Fuel", cr_account="Bank/Default",
    amount=1000, start=start_datetime, end=end_datetime, interval=1)
ops.add_activity(pay_delivery_costs)

pay_wages = BasicActivity("Pay Wages", dt_account="Expense/Wages",
    cr_account="Bank/Default", amount=10000, start=start_datetime, end=end_datetime,
    interval=1)
hr.add_activity(pay_wages)

# Run the model
model.run()

# Print the reports.
courier_company.gl.balance_sheet()
courier_company.gl.income_statement()
```

Table 3: Balance sheet report at the end of the courier business simulation.

| | | | |
|---|---|---|---|
| **Assets** | 186 409.63 | **Liabilities** | 0.00 |
| Bank/Default | 9407.63 | Long Term Borrowing/Capital Loan | 0.00 |
| Fixed Assets/Vehicle Asset | 177000.00 | | |
| | | **Owners' Equity** | 186 409.63 |
| | | Retained Income/Retained Earnings | 186 409.63 |
| **Total** | 186 409.63 | **Total** | 186 409.63 |

Table 4: Income statement report for the total duration of the courier business simulation.

| | **Debit** | **Credit** |
|---|---|---|
| **Revenues** | | 885 000.00 |
| Sales/Sales Delivery | | 885 000.00 |
| | | |
| **Cost of Sales** | 59 000.00 | |
| Cost of Sales/Fuel | 59 000.00 | |
| | | |
| **Gross Revenues** | | 826 000.00 |
| | | |
| **Expenses** | 639 590.37 | |
| Expense/Interest Expense | 49 590.37 | |
| Expense/Wages | 590 000.00 | |
| | | |
| **Net Income** | | 186 409.63 |

# Future Plans

Version 0.2.0 of `auxi` is presented in this article. It is a small, but useful start to a valuable set of tools to equip and empower metallurgical process engineers to solve problems in an increasingly challenging environment. The following aspects are planned for future versions:

**More calculation tools:** There are certain calculations that we need regularly. In pyrometallurgy, for example, it is often needed to calculate heat losses from exterior surfaces of a furnace. Equations and empirical correlations will be added to `auxi` to make this quick and reliable.

Similarly, other commonly used calculations will be added as these are identified. `auxi` users can submit requests or even contribute code themselves.

**Unit process models:** Models of screens, cyclones, spirals, and other mineral processing unit operations will be added to assist in building flow sheet models quickly and easily.

**Graphical user interface:** Although Python programming is powerful, not everyone will want to delve into code. We are planning to create a graphical user interface for `auxi` to allow complete novices to get started quickly. The code will always remain available, so that `auxi` does not become a black box.

The main focus areas are a flow sheet editor for process models, and a user interface for creating and configuring business models.

**Financial metrics:** Functions will be added to the financial module to calculate figures such as return on investment (ROI), internal rate of return (IRR), nett present value (NPV), and other metrics that enable analysis of business scenarios.

**Others:** There are other smaller aspects that we want to introduce and improve. These include integrating units of measure and improving data visualisation.

# Conclusion

In this article we presented a quick journey through a Python toolkit that helps with doing metallurgical calculations. It is perhaps not the most high-tech topic under discussion during the colloquium. We do, however, believe that it is about more than code and calculations.

`auxi` is an open-source toolkit that aims to stimulate sharing and collaboration, and drawing metallurgical process engineers in South Africa together. What has started out as something small, has the potential to grow into something that can make a significant contribution to how we beneficiate and add value to our mineral wealth. Technology is an important factor in the road ahead, but it will always be limited by and dependent on the capabilities of the people using it. We need to work together to succeed.

We encourage metallurgical process engineers and others who are interested to start using `auxi`, but more importantly, to start contributing through feedback, suggestions and even some code.

# References


Bale, C W et al. (2009). "FactSage thermochemical software and databases  recent developments". In: *Calphad* 33.2, pp. 295–311. ISSN: 0364-5916. DOI: http://dx.doi.org/10.1016/j.calphad.2008.09.009. URL: http://www.sciencedirect.com/science/article/pii/S0364591608000965.

Dinsdale, A. T. (1991). "SGTE data for pure elements". In: *Calphad* 15.4, pp. 317–425. ISSN: 03645916.

Free Software Foundation (2007). *GNU Lesser General Public License*. Version 3. Free Software Foundation. URL: http://www.gnu.org/licenses/lgpl-3.0.en.html (visited on 04/16/2016).

Python Software Foundation (2016). *The Python Language Reference*. URL: https://docs.python.org/3/reference/index.html (visited on 04/15/2016).

Rao, Y.K. (1985). *Stoichiometry and Thermodynamics of Metallurgical Processes*. Cambridge, UK: Cambridge University. ISBN: 0-521-25856-1.

Statistics South Africa (2016a). *Export and Import Value Indices - January 2016*. Statistics South Africa. URL: http://www.statssa.gov.za/publications/P01427/P01427January2016.pdf (visited on 04/15/2016).

— (2016b). *Gross domestic product - Fourth quarter 2015*. Statistics South Africa. URL: http://www.statssa.gov.za/publications/P0441/P04414thQuarter2015.pdf (visited on 04/15/2016).

Wikipedia (2016). *History of Python*. URL: https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=History_of_Python&id=707803140 (visited on 04/16/2016).