Acknowledgements

Lorem Ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum

Contents

Introduction

To provide a comprehensive background for the following chapters that focus on the interaction of human mesenchymal stromal cells (hMSCs) with multiple myeloma (MM) cells, this

Human Mesenchymal Stem/Stromal Cells

Explaining what a mesenchymal stromal cell (MSC) is, is not such an easy task as one might expect. MSCs are derived from multiple MSCs different sources, serve a wide array of functions and are always isolated as a heterogenous group of cells. This makes it particularly challenging to find a consensus on their exact definition, nomenclature, exact function and *in vivo* differentiation potential. Therefore, the most effective approach to describe hMSCs is to present their historical context.

hMSCs first gained popularity as a stem cell. Stem cells lay the foundation of multicellular organisms. Embryonic stem cells orchestrate the growth and patterning during embryonic development, while adult stem cells are responsible for regeneration during adulthood. The classical definition of a stem cell is that of a relatively undifferentiated cell that divides asymmetrically, producing another stem cell and a differentiated cell (Cooper, 2000; Shenghui et al., 2009). Because of their significance in biology and regenerative medicine, stem cells have become a prominent subject in modern research. Especially human mesenchymal stromal cells (hMSCs) have proven to be a promising candidate in this context (Ullah et al., 2015).

Mesenchyme first appears in embryonic development during gastrulation. There, cells that are committed to a mesodermal fate, lose their cell junctions and exit the epithelial layer in order to migrate freely. This process is called epithelial-mesenchymal transition (Tam & Beddington, 1987; Nowotschin & Hadjantonakis, 2010). Hence, the term mesenchyme describes non-epithelial embryonic tissue differentiating into mesodermal lineages such as bone, muscles and blood. Interestingly, it was shown nearly twenty years earlier that cells within adult bone marrow seemed to have mesenchymal properties as they were able to differentiate into bone tissue (A. J. Friedenstein et al., 1966; A. Friedenstein & Kuralesova, 1971; Bianco, 2014). This was the origin of the “mesengenic process”-hypothesis: This concept states that mesenchymal stem cells serve as progenitors for multiple mesodermal tissues (bone, cartilage, muscle, marrow stroma, tendon, fat, dermis and connective tissue) during both adulthood and embryonic development (A. Caplan, 1991; A. I. Caplan, 1994). The mesenchymal nature of these cells (termed bone marrow stromal cells: BMSCs) was confirmed later when they were shown to differentiate into adipocytic (fat) and chondrocytic (cartilage) lineages (Pittenger et al., 1999). Since then,

the term "mesenchymal stem cell" (MSC) has grown popular as an adult multipotent precursor to a couple of mesodermal tissues. hMSCs derived from bone marrow (hMSCs) were shown to differentiate into osteocytes, chondrocytes, adipocytes and cardiomyocytes (Gronthos et al., 1994; Muruganandan et al., 2009; Xu et al., 2004) Most impressively, these cells also exhibited ectodermal and endodermal differentiation potential, as they produced neuronal cells, pancreatic cells and hepatocytes (Barzilay et al., 2009; Wilkins et al., 2009; Gabr et al., 2013; Stock et al., 2014).

Furthermore, cultures with MSC properties can be established from "virtually every postnatal organs and tissues", and not just bone marrow (da Silva Meirelles et al., 2006). However, it has to be noted that hMSCs can differ greatly in their transcription profile and *in vivo* differentiation potential depending on which tissue they originated from (Jansen et al., 2010; Sacchetti et al., 2016).

Since "hMSCs' are a heterogenous group of cells, they were defined by their *in vitro* characteristics. A minimal set of criteria are the following (Dominici et al., 2006): First, hMSCs must be plastic adherent. Second, they must express or lack a set of specific surface antigens (positive for CD73, CD90, CD105; negative for CD45, CD34, CD11b, CD19). Third, hMSCs must differentiate to osteoblasts, adipocytes and chondroblasts *in vitro*. Together, hMSCs exhibit diverse differentiation potentials and can be isolated from multiple sources of the body. This offers great opportunity for regenerative medicine, if the particular hMSC-subtype is properly characterized.

Multiple Myeloma

Multiple myeloma arises from clonal expansion of malignant plasma cells in the bone marrow (BM). At diagnosis, myeloma cells have disseminated to multiple sites in the skeleton and, in some cases, to virtually any tissue (Rajkumar & Kumar, 2020; Bladé et al., 2022).

Myeloma-hMSC Interactions

Since plasma cells can not survive outside the bone marrow, MM cells also require survival signals for growth and disease progression. These signals are produced by the bone marrow microenvironment, including ECM, MSCs and ACs (Kibler et al., 1998; García-Ortiz et al., 2021).

Myeloma Bone Disease

Bone is a two-phase system in which the mineral phase provides the stiffness and the collagen fibers provide the ductility and ability to absorb energy (Viguet-Carrin et al., 2006). On a molecular level, bone tissue is composed of extracellular matrix (ECM) proteins that are calcified by hydroxyapatite crystals. This ECM consists mostly of collagen type I, but also components with major regulatory activity, such as fibronectin and proteoglycans that are essential for healthy bone physiology (Alcorta-Sevillano et al., 2020). Bone tissue is actively remodeled by bone-forming osteoblasts and bone-degrading osteoclasts. Osteoblasts are derived from mesenchymal stromal cells (MSCs) that reside in the bone marrow (A. J. Friedenstein et al., 1966; Pittenger et al., 1999). MSCs also give rise to adipocytes (ACs) to form Bone Marrow Adipose Tissue (BMAT), which can account for up to 70% of bone marrow volume (Fazeli et al., 2013).

MM indirectly degrades bone tissue by stimulating osteoclasts and inhibiting osteoblast differentiation, which leads to MM-related bone disease (MBD) (Glavey et al., 2017). MBD is present in 80% of patients at diagnosis and is characterized by osteolytic lesions, osteopenia and pathological fractures (Terpos et al., 2018).

Dissemination of Myeloma Cells

dissemination is still widely unclear - multistep process - invasion, intravasation, intravascular arrest, extravasation, colonization - overcome adhesion, retention, and dependency on the BM microenvironment - loss of adhesion factors such as CD138

The Increasing Role of Software in Biomedicine

In the last decades, biosciences have made significant progress in generating vast amounts of data in shorter time spans (Yang et al., 2017). Here, it is argued that research reliant on software more than ever. Moreso,

Modern methods in molecular biology, biochemistry, and biomedicine, such as next-generation sequencing, mass spectrometry, and high-throughput screening, generate large volumes of data that require sophisticated software tools for analysis. For instance, bioinformatics software is essential for analyzing genomic and proteomic data, while image analysis software is routinely used in microscopy.

Moreover, the rise of systems biology and integrative biomedicine, which aim to understand biological systems as a whole, has led to the development of complex computational models and simulation software. These tools are used to integrate and analyze diverse data types, from molecular to physiological data, and to predict the behavior of biological systems.

In addition, software plays a crucial role in the management and sharing of biomedical data. Databases and data repositories are essential for storing, retrieving, and sharing data, while data standards and ontologies, which are often implemented as software libraries, are used to ensure that data is interoperable and reusable.

Given this landscape, it is clear that researchers in the biosciences are confronted with complex software on a daily basis. Therefore, there is a growing need for researchers to acquire computational skills. Learning a programming language like Python can greatly benefit researchers by enabling them to automate tasks, analyze data more efficiently, and develop their own tools. This not only increases productivity but also fosters reproducibility and open science.

In conclusion, the increasing role of software in biomedicine underscores the importance of computational skills for modern researchers. As the field continues to evolve, the ability to work with software will become even more critical.

- RNAseq - single cell rnaseq - sequence

Artificial intelligence (AI) has been a game changer in the field of biomedicine. The early development of AI itself was driven by radiology, where it was designed to detect pathologies in medical images.

Code Quality Ensures Scientific Reproducibility

A main reason to write software is to define re-usable instructions for task automation (Narzt et al., 1998). However, the complexity of the code makes it prone to errors and can prevent

usage by persons other than the author himself. This is a problem for the general scientific community, as the software is often the only way to reproduce the results of a study (Sandve et al., 2013). Hence, modern journals aim to enforce standards to software development, including software written and used by biological researchers (Smith et al., 2018). Here, we provide a brief overview of the standards utilized by `plotastic` that to ensure its reliability and reproducibility by the scientific community.

Modern software development is a long-term commitment of maintaining and improving code after initial release (Boswell & Foucher, 2011). Hence, it is good practice to write the software such that it is scalable, maintainable and usable. Scalability or, to be precise, structural scalability means that the software can easily be expanded with new features without major modifications to its architecture (Bondi, 2000). This is achieved by writing the software in a modular fashion, where each module is responsible for a single function. Maintainability means that the software can easily be fixed from bugs and adapted to new requirements (Kazman et al., 2020). This is achieved by writing the code in a clear and readable manner, and by writing tests that ensure that the code works as expected (Boswell & Foucher, 2011). Usability is hard to define (Brooke, 1996), yet one can consider a software as usable if the commands have intuitive names and if the software's manual, termed "documentation", is up-to-date and easy to understand for new users with minimal coding experience. A software package that has not received an update for a long time (approx. one year) could be considered abandoned. Abandoned software is unlikely to be fully functional, since it relies on other software (dependencies) that has changed in functionality or introduce bugs that were not expected by the developers of all dependencies. Together, software that's scalable, maintainable and usable requires continuous changes to its codebase. There are best practices that standardize the continuous change of the codebase, including version control, continuous integration (often referred to as CI), and software testing.

Version control is a system that records changes to the codebase line by line, allowing the documentation of the history of the codebase, including who made which changes and when. This is required to isolate new and experimental features into newer versions and away from the stable version that's known to work. The most popular version control system is Git, which is considered the industry standard for software development (Chacon & Straub, 2024). Git can use GitHub.com as a platform to store and host codebases in the form of software repositories. GitHub's most famous feature is called "pull request". A pull request is a request from anyone registered on GitHub to include their changes to the codebase (as in "please pull this into your main code"). One could see pull requests as the identifying feature of the open source community, since it exposes the codebase to potentially thousands of independent developers, reaching a workforce that is impossible to achieve with closed source models used by paid software companies.

Continuous integration (CI) is a software development practice in which developers integrate code changes into a shared repository several times a day (Duvall et al., 2007). Each integration triggers the test suite, aiming to detect errors as soon as possible. The test suite includes building the software, setting up an environment for the software to run and then executing the programmed tests, ensuring that the software runs as a whole. Continuous integration is often used together with software branches. Branches are independent copies of the codebase that are meant to be merged back into the original code once the changes are finished. Since branches accumulate multiple changes over time, this can lead to minor incompatibilities between the branches of all developers (integration conflicts), which is something that CI helps to prevent.

Continuous integration especially relies on a thorough software testing suite. Software testing is the practice of writing code that checks if the codebase works as expected (Myers et al., 2011). The main type of software testing is unit testing, which tests the smallest units of the codebase (functions and classes) in isolation (Listing 1).

Listing 1: Example of an arbitrary python function and its respective unit test function. The first function simply returns the number 5. The second function tests if the first function indeed returns the number 5. The test function is named with the prefix "`test_`" and is placed in a file that ends with the suffix "`_test.py`". The test function is executed by the testing framework `pytest`. Note that code after "#" is considered a comment and won't be executed.

```
# Define a function called "give_me_five" that returns the number 5
def give_me_five():
    return 5
# Define a test function asserting that "give_me_five" returns 5
def test_give_me_five():
    assert give_me_five() == 5
```

The quality of the software testing suite is measured by the code coverage, the precision of the tests, and the number of test-cases that are checked. The code coverage is the percentage of the codebase that is called by the testing functions, which should be as close to 100% as possible, although it does not measure how well the code is tested. The precision of the test is not a measurable quantity, but it represents if the tests truly checks if the code works as expected. The number of test-cases is the number of different scenarios that are checked by the testing functions, for example testing every possible option or combinations of options for functions that have multiple options. The most popular software testing framework for python is `pytest`, which is utilized by `plotastic` (Krekel et al., 2004).

Python as a Programming Language

Here, we provide a general overview of the python programming language, explaining terms like *"type"*, *"method"*, etc., in order to prepare readers without prior programming experience

for the following chapters. We also describe the design principles of python to lay out the key concepts that differentiate python compared to other programming languages.

Languages such as python are considered *"high-level"*, which means that it is designed to be easy to read and write, but also independent of hardware by hiding (*"abstracting"*) underlying details (The Python Language Reference, n.d.). A key principle of python is the emphasis on implementing a syntax that is concise and close to human language (Listing 2, Listing 3).

Listing 2: Example of readable python code. This one-line code returns the words (string) `'Hello, World!'` when executed. The command is straightforward and easy to understand.

```python
print("Hello, World!")
# Output: Hello, World!
```

Listing 3: Example of less readable code written in the low-level programming language C. This code is doing exactly the same as the python code in Listing 2. The command is harder to understand because more steps are needed to access the same functionality, including the definition of a function

```c
#include <stdio.h>
int main() {
    printf("Hello, World!");
    return 0;
}
// Output: Hello, World!
```

Furthermore, python is an *interpreted* language, which means that the code is executed line by line. This makes coding easier because the programmer can see the results of the code immediately after writing it, and error messages point to the exact line where the error occurred. This is in contrast to *compiled* languages, where the code has to be compiled into machine code before it can be executed. The advantage of compiled languages is that the code runs faster, because the machine code is optimized for the hardware.

Python automates tasks that would otherwise require an advanced understanding of computer hardware, like the need for manual allocation of memory space. This is achieved by using a technique called *"garbage collection"*, which automatically frees memory space that is no longer needed by the program. This is a feature that is not present in low-level programming languages like C or C++, that were designed to maximize control over hardware.

Another hallmark of python is its *dynamic typing system*. In python the type is inferred automatically during code execution (Listing 4). This is in contrast to *statically* typed languages like C, where the type of a variable has to be declared explicitly and cannot be changed during code execution (Listing 5) (The Python Language Reference, n.d.).

Dynamic typing makes python a very beginner-friendly language, since one does not have to keep track of the type of each variable. However, this also makes python a slower language,

Listing 4: Example of dynamic typing in python. The variable "a" is assigned the value 5, which is an integer. The variable "a" is then assigned the value "Hello, World!", which is a string. Python allows Note that code after "#" is considered a comment and won't be executed.

```
a = 5  # Type integer
a = 5.0  # Type float
a = 'Hello, World!'  # Type string
a = True  # Type boolean
a = False  # Type boolean
a = [1, 2, 3]  # Type list of integers
a = {'name': 'Regina'}  # Type dictionary
```

Listing 5: Example of static typing in C. The variable "a" is declared as an integer, and can only store integers. The variable "a" is then assigned the value 5, which is an integer. The variable "a" is then assigned the value 'Hello, World!', which is a string. This results in a compilation error, because the variable "a" can only store integers.

```
int a;  // Declare type as integer
a = 5;
a = 'Hello, World!';  // Compilation error!
```

because the interpreter has to check the type of each variable during code execution. Also, developing code with dynamic typing systems is prone to introducing bugs ("type errors"), because it allows unexperienced developers to convert variables from one type to another without noticing, leading to unexpected behavior. Hence, larger python projects require disciplined adherence to programming conventions. One such convention is *type hinting*, which is a way to explicitly note the type of a variable. Type hinting does not have an effect on the code, but it makes the code more readable and understandable for other developers, and allows for development environments to detect type errors before execution. (Listing 6) (van Rossum et al., 2014).

Listing 6: Example of type hints used in python. Explicitly stating the type of the variable is optional and does not change the behavior of the code as shown in Listing 4.

```
a: int = 5
a: str = 'Hello, World!'
```

Python supports both functional and object-oriented programming paradigms. In functional programming, the code is written in a way that the program is a sequence of function calls, where each function call returns a value that is used in the next function call (Listing 7). This approach is useful when multiple actions have to be performed on the same data and the structure of the data is relatively simple, for example a string of a gene.

When the data itself gains in complexity, for example when associating the promotor with a gene sequence, an object-oriented approach is more suitable (Listing 8). Object-oriented programming is a programming paradigm that uses objects and classes. An object is a collection of both data and functions, and a class is a blueprint for creating objects. The data of an object

Listing 7: Example of functional programming in Python. The code defines a function called "find_restriction_site" that finds the position of a restriction site in a gene. The function "cut" uses the function "find_restriction_site" to cut the gene at the restriction site.

```python
def find_restriction_site(gene: str):
    return gene.find('GCGC')

def cut(gene: str):
    position = find_restriction_site(gene)
    return gene[position:]

gene1 = 'TGAGCTGAGCTGATGCGCTATATTTAGGCG'
gene1_cut = cut(gene1)
print(gene1_cut)
# Output: GCGCTATATTTAGGCG
```

is stored as attributes. Functions that are associated with an object are called methods.

Listing 8: Example of object oriented programming in python. The class is called "Gene" and has four methods, "__init__", "find_promotor", "find_restriction_site" and "cut". The function "__init__" is called when creating ("initializing") an object, which fills the object with user-defined data. The parameter "self" is used to reference the object itself internally. "find_promotor" is a method that finds the position of the promotor in the gene and is called during object initialization.

```python
class Gene:
    def __init__(self, sequence: str):
        self.sequence: str = sequence # Save sequence as attribute
        self.promotor: str = self.find_promotor()
    def find_promotor(self):
        return self.sequence.find('TATA')
    def find_restriction_site(self):
        return self.sequence.find('GCGC')
    def cut(self):
        position = self.find_restriction_site()
        return self.sequence[position:]

gene1 = Gene(sequence='TGAGCTGAGCTGATGCGCTATATTTAGGCG') # Create object
gene1_cut = gene1.cut() # Call the method cut
print(gene1_cut) # Show result
# Output: GCGCTATATTTAGGCG
```

A major benefit of using an object oriented versus a functional approach is that the data itself programmable, enabling the programmer to define the behavior of the data itself through methods. This is achieved by using the keyword "`self`" to reference the object itself inside the class. For example, one could extend the class "`Gene`" with a method that finds the promotor of the gene and stores it as an attribute (Listing 8).

When designing software, both functional and object oriented programming can be used together, where object oriented programming is often used to design the program's overall

architecture, and functional programming is used to implement the algorithms of the program's features. This allows for scalability of the software, as every single class is extended through the addition of new methods. Furthermore, classes can be expanded in their functionalities through inheritance (Listing 9).

Listing 9: Example of inheritance in python. The class "mRNA" inherits from the class "Gene". The class "mRNA" has two methods, "__init__" and "find_stopcodon". The method "find_stopcodon" finds the position of stop codons.

```python
# Define a class called mRNA inheriting from the class Gene
class mRNA(Gene):
    def __init__(self, sequence: str):
        super().__init__(sequence)  # Get attributes from parent class
        self.sequence.replace('T', 'U')  # Replace thymine with uracil
    def find_stopcodons(self):
        return self.sequence.find('UGA')

mrna1 = mRNA(sequence='TGAGCTGAGCTGATGCGCTATATTTAGGCG') # Create object
print(mrna1.find_stopcodons())  # Call the method translate
# Output: [0, 5, 10]
```

Inheritance is a feature of object-oriented programming that allows a class to access every attribute and method of a parent class. For example, one could extend the class "`Gene`" with a class "`mRNA`", by writing a class "`mRNA`" that inherits from the class "`Gene`".

Together, python is not just beginner-friendly, but also well respected for its ease n development, which is why it is widely used in professional settings for web development, data analysis, machine learning, and more.

Data Science with Python

the ease of use has made python a very popular language (Rayhan & Gross, 2023)

Like any other programming language, python alone does not provide specialized tools like those used for data analysis (The Python Language Reference, n.d.). However, python was designed to be extended by packages developed by its users. A python package consists of multiple python modules, where each module is a text-file with a `.py` ending containing python code. Famous examples of such packages are `pytorch` and `tensorflows`, that are used to build models of artificial intelligence, including ChatGPT (Paszke et al., 2019; Abadi et al., 2016; Radford et al., 2019). Here, we outlay the most important packages used for `plotastic`.

Interactive Python - Jupyter

Python overcame the issues of interpreted language by utilizing Code written in C numpy: - Acceleration, - SIMD instructions

Tabular operations - pandas

Data visualization - matplotlib - seaborn

Inferential Statistics - pingouin

AI: - pytorch and tensorflow - example: VGG19 is just a few lines of code (**??**) asdfdf

Convolutional Neural Networks

This work greatly benefited from the use of convolutional neural networks provided by the Zeiss ZEN software. Here, we provide a brief

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … Zheng, X. (2016, March). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (No. arXiv:1603.04467). arXiv. Retrieved 2024-03-07, from http://arxiv.org/abs/1603.04467 doi: 10.48550/arXiv.1603.04467

Alcorta-Sevillano, N., Macías, I., Infante, A., & Rodríguez, C. I. (2020, December). Deciphering the Relevance of Bone ECM Signaling. Cells, 9(12), 2630. Retrieved 2023-12-20, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7762413/ doi: 10.3390/cells9122630

Barzilay, R., Ben-Zur, T., Bulvik, S., Melamed, E., & Offen, D. (2009, May). Lentiviral delivery of LMX1a enhances dopaminergic phenotype in differentiated human bone marrow mesenchymal stem cells. Stem cells and development, 18(4), 591–601. doi: 10.1089/scd.2008.0138

Bianco, P. (2014). "Mesenchymal" stem cells. Annual review of cell and developmental biology, 30, 677–704. doi: 10.1146/annurev-cellbio-100913-013132

Bladé, J., Beksac, M., Caers, J., Jurczyszyn, A., von Lilienfeld-Toal, M., Moreau, P., … Richardson, P. (2022, March). Extramedullary disease in multiple myeloma: A systematic literature review. Blood Cancer Journal, 12(3), 1–10. Retrieved 2023-03-24, from https://www.nature.com/articles/s41408-022-00643-3 doi: 10.1038/s41408-022-00643-3

Bondi, A. B. (2000, September). Characteristics of scalability and their impact on performance. In Proceedings of the 2nd international workshop on Software and performance (pp. 195–203). New York, NY, USA: Association for Computing Machinery. Retrieved 2024-03-07, from https://dl.acm.org/doi/10.1145/350391.350432 doi: 10.1145/350391.350432

Boswell, D., & Foucher, T. (2011). The Art of Readable Code: Simple and Practical Techniques for Writing Better Code. "O'Reilly Media, Inc.".

Brooke, J. (1996, January). SUS – a quick and dirty usability scale. In (pp. 189–194).

Caplan, A. (1991). Mesenchymal stem cells. Journal of orthopaedic research : official publication of the Orthopaedic Research Society, 9(5), 641–50. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/1870029 doi: 10.1002/jor.1100090504

Caplan, A. I. (1994, July). The mesengenic process. Clinics in plastic surgery, 21(3), 429–435.

Chacon, S., & Straub, B. (2024, March). Git - Book. Retrieved 2024-03-07, from https://git-scm.com/book/de/v2

Cooper, G. M. (2000). The Cell: A Molecular Approach. 2nd Edition. Sinauer Associates, Proliferation in Development and Differentiation. Retrieved from https://www.ncbi.nlm.nih.gov/books/NBK9906/

da Silva Meirelles, L., Chagastelles, P. C., & Nardi, N. B. (2006, June). Mesenchymal stem cells reside in virtually all post-natal organs and tissues. Journal of cell science, 119(Pt 11), 2204–2213. doi: 10.1242/jcs.02932

Dominici, M., Le Blanc, K., Mueller, I., Slaper-Cortenbach, I., Marini, F., Krause, D., … Horwitz, E. (2006). Minimal criteria for defining multipotent mesenchymal stromal cells. The International Society for Cellular Therapy position statement. Cytotherapy, 8(4), 315–317. doi: 10.1080/14653240600855905

Duvall, P., Matyas, S., & Glover, A. (2007). Continuous integration: Improving software quality and reducing risk. Pearson Education. Retrieved from https://books.google.de/books?id=PV9qfEdv9L0C

Fazeli, P. K., Horowitz, M. C., MacDougald, O. A., Scheller, E. L., Rodeheffer, M. S., Rosen, C. J., & Klibanski, A. (2013, March). Marrow Fat and Bone—New Perspectives. The Journal of Clinical Endocrinology and Metabolism, 98(3), 935–945. Retrieved 2023-12-20, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3590487/ doi: 10.1210/jc.2012-3634

Friedenstein, A., & Kuralesova, A. I. (1971, August). Osteogenic precursor cells of bone marrow in radiation chimeras. Transplantation, 12(2), 99–108.

Friedenstein, A. J., Piatetzky-Shapiro, I. I., & Petrakova, K. V. (1966, December). Osteogenesis in transplants of bone marrow cells. Journal of embryology and experimental morphology, 16(3), 381–390.

Gabr, M. M., Zakaria, M. M., Refaie, A. F., Ismail, A. M., Abou-El-Mahasen, M. A., Ashamallah, S. A., … Ghoneim, M. A. (2013). Insulin-producing cells from adult human bone marrow mesenchymal stem cells control streptozotocin-induced diabetes in nude mice. Cell transplantation, 22(1), 133–145. doi: 10.3727/096368912X647162

García-Ortiz, A., Rodríguez-García, Y., Encinas, J., Maroto-Martín, E., Castellano, E., Teixidó, J., & Martínez-López, J. (2021, January). The Role of Tumor Microenvironment in Multiple Myeloma Development and Progression. Cancers, 13(2). Retrieved 2021-02-02, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7827690/ doi: 10.3390/cancers13020217

Glavey, S. V., Naba, A., Manier, S., Clauser, K., Tahri, S., Park, J., … Ghobrial, I. M. (2017, November). Proteomic characterization of human multiple myeloma bone marrow extracellular matrix. Leukemia, 31(11), 2426–2434. Retrieved 2023-09-05, from https://www.nature.com/articles/leu2017102 doi: 10.1038/leu.2017.102

Gronthos, S., Graves, S. E., Ohta, S., & Simmons, P. J. (1994, December). The STRO-1+ fraction of adult human bone marrow contains the osteogenic precursors. Blood, 84(12), 4164–4173.

Jansen, B. J. H., Gilissen, C., Roelofs, H., Schaap-Oziemlak, A., Veltman, J. A., Raymakers, R. A. P., … Adema, G. J. (2010, April). Functional differences between mesenchymal stem cell populations are reflected by their transcriptome. Stem cells and development, 19(4), 481–490. doi: 10.1089/scd.2009.0288

Kazman, R., Bianco, P., Ivers, J., & Klein, J. (2020, December). Maintainability (Report). Carnegie Mellon University. Retrieved 2024-03-07, from https://kilthub.cmu.edu/articles/report/Maintainability/12954908/1 doi: 10.1184/R1/12954908.v1

Kibler, C., Schermutzki, F., Waller, H. D., Timpl, R., Müller, C. A., & Klein, G. (1998, June). Adhesive interactions of human multiple myeloma cell lines with different extracellular matrix molecules. Cell Adhesion and Communication, 5(4), 307–323. doi: 10.3109/15419069809040300

Krekel, H., Oliveira, B., Pfannschmidt, R., Bruynooghe, F., Laugher, B., & Bruhin, F. (2004). Pytest. Retrieved from https://github.com/pytest-dev/pytest

Muruganandan, S., Roman, A. A., & Sinal, C. J. (2009, January). Adipocyte differentiation of bone marrow-derived mesenchymal stem cells: Cross talk with the osteoblastogenic program. Cellular and molecular life sciences : CMLS, 66(2), 236–253. doi: 10.1007/s00018-008-8429-z

Myers, G. J., Sandler, C., & Badgett, T. (2011). The art of software testing (3rd ed.). Wiley Publishing. Retrieved from https://malenezi.github.io/malenezi/SE401/Books/114-the-art-of-software-testing-3-edition.pdf

Narzt, W., Pichler, J., Pirklbauer, K., & Zwinz, M. (1998, January). A Reusability Concept for Process Automation Software..

Nowotschin, S., & Hadjantonakis, A.-K. (2010, August). Cellular dynamics in the early mouse embryo: From axis formation to gastrulation. Current opinion in genetics & development, 20(4), 420–427. doi: 10.1016/j.gde.2010.05.008

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019, December). PyTorch: An Imperative Style, High-Performance Deep Learning Library (No. arXiv:1912.01703). arXiv. Retrieved 2024-03-07, from http://arxiv.org/abs/1912.01703 doi: 10.48550/arXiv.1912.01703

Pittenger, M. F., Mackay, A. M., Beck, S. C., Jaiswal, R. K., Douglas, R., Mosca, J. D., … Marshak, D. R. (1999). Multilineage Potential of Adult Human Mesenchymal Stem Cells. , 284(April), 143–148. doi: 10.1126/science.284.5411.143

The Python Language Reference. (n.d.). Retrieved 2024-03-07, from https://docs.python.org/3/reference/

index.html

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners.. Retrieved 2024-03-07, from https://www.semanticscholar.org/paper/Language-Models-are-Unsupervised-Multitask-Learners-Radford-Wu/9405cc0d6169988371b2755e573cc28650d14dfe

Rajkumar, S. V., & Kumar, S. (2020, September). Multiple myeloma current treatment algorithms. Blood Cancer Journal, 10(9), 94. Retrieved 2023-07-03, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7523011/ doi: 10.1038/s41408-020-00359-2

Rayhan, A., & Gross, D. (2023). The Rise of Python: A Survey of Recent Research. doi: 10.13140/RG.2.2.27388.92809

Sacchetti, B., Funari, A., Remoli, C., Giannicola, G., Kogler, G., Liedtke, S., … Bianco, P. (2016). No identical "mesenchymal stem cells" at different times and sites: Human committed progenitors of distinct origin and differentiation potential are incorporated as adventitial cells in microvessels. Stem Cell Reports, 6(6), 897–913. Retrieved from http://dx.doi.org/10.1016/j.stemcr.2016.05.011 doi: 10.1016/j.stemcr.2016.05.011

Sandve, G. K., Nekrutenko, A., Taylor, J., & Hovig, E. (2013, October). Ten Simple Rules for Reproducible Computational Research. PLoS Computational Biology, 9(10), e1003285. Retrieved 2024-03-07, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3812051/ doi: 10.1371/journal.pcbi.1003285

Shenghui, H., Nakada, D., & Morrison, S. J. (2009). Mechanisms of Stem Cell Self-Renewal. Annual Review of Cell and Developmental Biology, 25(1), 377–406. Retrieved from https://doi.org/10.1146/annurev.cellbio.042308.113248 doi: 10.1146/annurev.cellbio.042308.113248

Smith, A. M., Niemeyer, K. E., Katz, D. S., Barba, L. A., Githinji, G., Gymrek, M., … Vanderplas, J. T. (2018). Journal of Open Source Software (JOSS): Design and first-year review. PeerJ Preprints, 4, e147. doi: 10.7717/peerj-cs.147

Stock, P., Bruckner, S., Winkler, S., Dollinger, M. M., & Christ, B. (2014, April). Human bone marrow mesenchymal stem cell-derived hepatocytes improve the mouse liver after acute acetaminophen intoxication by preventing progress of injury. International journal of molecular sciences, 15(4), 7004–7028. doi: 10.3390/ijms15047004

Tam, P. P., & Beddington, R. S. (1987, January). The formation of mesodermal tissues in the mouse embryo during gastrulation and early organogenesis. Development (Cambridge, England), 99(1), 109–126.

Terpos, E., Ntanasis-Stathopoulos, I., Gavriatopoulou, M., & Dimopoulos, M. A. (2018, January). Pathogenesis of bone disease in multiple myeloma: From bench to bedside. Blood Cancer Journal, 8(1), 7. doi: 10.1038/s41408-017-0037-4

Ullah, I., Subbarao, R. B., & Rho, G. J. (2015). Human mesenchymal stem cells - current trends and future prospective Bioscience Reports.
doi: 10.1042/BSR20150025

van Rossum, G., Lehtosalo, J., & Langa. (2014). PEP 484 – Type Hints | peps.python.org. Retrieved 2024-03-08, from https://peps.python.org/pep-0484/

Viguet-Carrin, S., Garnero, P., & Delmas, P. D. (2006, March). The role of collagen in bone strength. Osteoporosis International, 17(3), 319–336. Retrieved 2023-12-20, from https://doi.org/10.1007/s00198-005-2035-9 doi: 10.1007/s00198-005-2035-9

Wilkins, A., Kemp, K., Ginty, M., Hares, K., Mallam, E., & Scolding, N. (2009, July). Human bone marrow-derived mesenchymal stem cells secrete brain-derived neurotrophic factor which promotes neuronal survival in vitro. Stem cell research, 3(1), 63–70. doi: 10.1016/j.scr.2009.02.006

Xu, W., Zhang, X., Qian, H., Zhu, W., Sun, X., Hu, J., … Chen, Y. (2004, July). Mesenchymal stem cells from

adult human bone marrow differentiate into a cardiomyocyte phenotype in vitro. Experimental biology and medicine (Maywood, N.J.), 229(7), 623–631.

Yang, A., Troup, M., & Ho, J. W. (2017, July). Scalability and Validation of Big Data Bioinformatics Software. Computational and Structural Biotechnology Journal, 15, 379–386. Retrieved 2024-03-07, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5537105/ doi: 10.1016/j.csbj.2017.07.002