Session 1: Introduction to Python for Healthcare Researchers

Why Python?

1. **Ease of Learning:** Python's simple and readable syntax
2. **Versatility and Efficiency:** Python's rich ecosystem of libraries and tools make it extremely versatile. Packages such as pandas for data manipulation, matplotlib for data visualization, and scikit-learn for machine learning make Python an efficient tool for health data science.
3. **Reproducibility:** Python's support for scripting allows for reproducibility
4. **Interdisciplinary Usage:** Python is not exclusive to programmers.

Running Python

There are two primary ways to run Python code

1. **Interpreter mode** - type python commands directly into the terminal
2. **Scripting mode** - write code in a file with a .py extension and run it using Python

Interpreter mode

```
python
```

```
>>>
```

```
>>> print("Hello, world!")
```

You will get:

```
 Hello, world!
```

Scripting mode

`hello.py`

```
print("Hello, world!")
```

We run this:

```
python hello.py
```

And we get:

```
Hello, world!
```

✏️ Exercise 1

1. Create a python file called `lesson1.py` on your computer using your text editor.
2. Add a print statement and then run this file from the command line.

✏️ Exercise 1 Solution

```
print("Hello, from lesson1.py")
```

You should get printed to the command line:

```
Hello, from lesson1.py
```

Variables

Variables are used to store values. They can be used to store different types of data, such as numbers, text, or logical values.

```
drug_name = "Aspirin"
quantity = 10
concentration = 2.5
is_prescription_only = True
```

Using variables

```
print(drug_name)
```

And we get:

```
Aspirin
```

We can insert them into strings using f-strings:

```
print(f"{drug_name} is a drug.")
```

And we get:

```
Aspirin is a drug.
```

Operations on variables

```
print(quantity * concentration)
```

And we get:

```
25.0
```

Saving the output of an operation to a variable

```
total_dose = quantity * concentration
print(total_dose)
```

And we get:

```
25.0
```

✏️ Exercise 2

> Modify your `lesson1.py` file to include the following:
>
> 1. Create a variable called `drug_name` and assign it the value `"Aspirin"`.
> 2. Create a variable called `quantity` and assign it the value `10`.
> 3. Create a variable called `text` and use your `drug_name` and `quantity` variables to create the following string: `"Aspirin is a drug that comes in a pack of 10."`.

✏️ Exercise 2 Solution

```
drug_name = "Aspirin"
quantity = 10
text = f"{drug_name} is a drug that comes in a pack of {quantity}."
print(text)
```

You should get printed to the command line:

```
Aspirin is a drug that comes in a pack of 10.
```

Comparison operators

```
quantity = 10
print(quantity == 10)
```

And we get:

```
True
```

Comparison operators

We can use the following operators to compare variables:

- `==` : Equal to
- `!=` : Not equal to
- `>` : Greater than
- `<` : Less than
- `>=` : Greater than or equal to
- `<=` : Less than or equal to

Types

Python has several built-in types. We have already come across several. The most commonly used ones are:

1. **Integers:** Whole numbers, e.g., 3, -2, 10.
2. **Floats:** Real numbers, e.g., 3.2, -0.9, 3.14.
3. **Strings:** Text, e.g., "Aspirin", 'Paracetamol'.
4. **Booleans:** Logical values, either True or False.

```
drug_name = "Aspirin" # string
quantity = 10 # integer
concentration = 2.5 # float
is_prescription_only = True # boolean`
```

Checking a type

We can check the type of a variable using the `type` function:

```
>>> type("aspirin")
```

or from a python file:

```
print(type("aspirin"))
```

And we get:

```
<class 'str'>
```

Converting between types

We can convert some types to others using the following functions but we may lose information in the process:

```
int(3.14) # 3
float(3) # 3.0
str(3) # "3"
bool(0) # False
```

Check type of a variable

```
concentration = 2.5
type(concentration) # float
```

Functions

Functions are reusable pieces of code. They take in inputs, perform some actions, and return a result.

```
def add_two_numbers(x, y):
    return x + y
```

We can then use this function:

```
add_two_numbers(3, 5) # 8
```

Saving the output of a function to a variable

```
total = add_two_numbers(3, 5)
print(total) # 8
```

Notes about functions

1. They start with the def keyword.
2. They have a name, in this case, `add_two_numbers`.
3. They have arguments, in this case, x and y. These are the inputs to the function. You can have as many arguments as you like, including none.
4. They have a return statement.

✏️ Exercise 3

1. Create a new python file called `drug_group1.py`.
2. Create a function called `determine_aspirin_group` that takes in a drug name and returns the drug group.
3. Test your function by calling it with the following drug name: "Aspirin".

✏️ Exercise 3 Solution

```
def determine_aspirin_group(drug_name):
    return "NSAID"
```

We can now test our function:

```
print(determine_aspirin_group("Aspirin")) # NSAID
```

Control Flow

If/else statements allow us to perform different actions depending on the value of a variable.

```
if drug_name == "Aspirin":
    return "NSAID"
else:
    return "Unknown"
```

Control Flow - Multiple conditions

We can also use elif statements to check multiple conditions:

```
if drug_name == "Aspirin":
    return "NSAID"
elif drug_name == "Paracetamol":
    return "Analgesic"
else:
    return "Unknown"
```

Control Flow - Adding complexity

We can build up more complex conditions using the following operators:

- `and` : Both conditions must be true.
- `or` : At least one condition must be true.
- `not` : The condition must be false.

✏️ Exercise 4

1. Create a function called `get_drug_group` that takes in a drug name and returns the drug group.

Drug Name	Drug Group
aspirin	NSAID
paracetamol	Analgesic
propranolol	Beta Blocker
ibuprofen	NSAID

2. Test your function by calling it with the following drug names: "Aspirin", "Paracetamol", "Caffeine".

✏️ Exercise 4 Solution

```
def get_drug_group(drug_name):
    if drug_name == "aspirin":
        return "NSAID"
    elif drug_name == "paracetamol":
        return "Analgesic"
    elif drug_name == "propranolol":
        return "Beta Blocker"
    elif drug_name == "ibuprofen":
        return "NSAID"
    else:
        return "Unknown"
```

✏️ Exercise 4 Solution

We can now test our function:

```
print(get_drug_group("aspirin")) # NSAID
print(get_drug_group("paracetamol")) # Analgesic
print(get_drug_group("caffeine")) # Unknown
```

Sets are similar as lists but they are unmutable and cannot contain duplicates.

```
drug_set = {"Aspirin", "Paracetamol", "Ibuprofen", "Aspirin"}
print(drug_set) # {"Aspirin", "Paracetamol", "Ibuprofen"} - duplicates are removed
```

Dictionaries

```
drug_dict = {"Aspirin": "NSAID", "Paracetamol": "Analgesic", "Ibuprofen": "NSAID"}
drug_dict["Aspirin"] # "NSAID" - gets the value for the given key
drug_dict['Caffeine'] = "Other" # {"Aspirin": "NSAID", "Paracetamol": "Analgesic", "Ibuprofen": "NSAID", "Caffeine": "Other"} - adds a new key-value pair
drug_dict.pop("Aspirin") # {"Paracetamol": "Analgesic", "Ibuprofen": "NSAID", "Caffeine": "Stimulant"} - removes a key-value pair
drug_dict['Caffeine'] = "Stimulant" # {"Paracetamol": "Analgesic", "Ibuprofen": "NSAID", "Caffeine": "Stimulant"} - updates a key-value pair
```

> ✏️ **Exercise 5**
>
> Modify your `get_drug_group` function to use a dictionary instead of if statements.

▶ Answer

7. Loops

Looping allows us to perform the same action multiple times. There are two types of loops in Python: for loops and while loops. For now, we'll focus on for loops.