Variables in Python

Variables

In this section we'll talk about the concept of **variables**, how to name them properly, overwrite, delete and keep track of them

TOPIC WE'LL COVER	GOALS FOR THIS SECTION
Variable Assignment	• Learn to assign variables in Python
Overwriting & Deleting	• Understand the behavior for overwriting variables
Naming Conventions	• Learn the rules & best practices for naming variables
Tracking Variables	

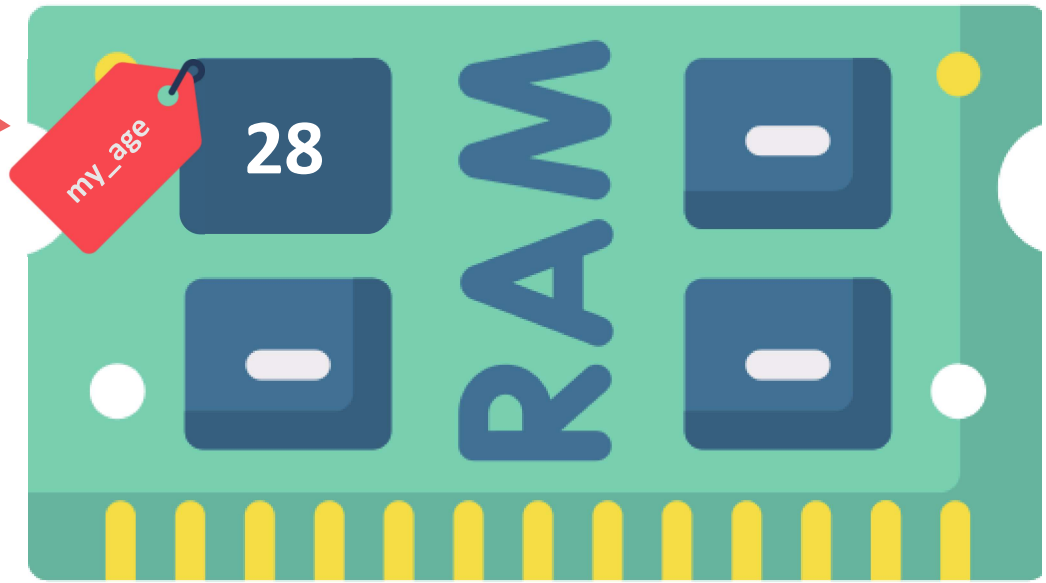
“Assigning” in Programming

In Computing: Assign → To put a value in a particular position in the **memory** (RAM) of a computer

```
my_age = 28
```

= is NOT “equality” in Math

Run




```
print(my_age)
```

28

```
print(my_age + 10)
```

38

Variable Assignment

Variables are containers used to label and store values for future use

- They can store **any** Python data type (and even call to function!)

```
variable_name = value
```

Proper variable names

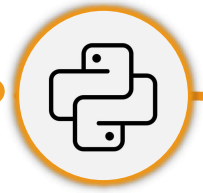
Examples:
- my_age
- total_price
- username
- ages_lst

Values assigned to the variables

Examples:
- 28
- 350.45
- 'Ahmad'
- [25, 20, 32]

DevTip: It's recommended **to add space on both sides** of the **equal** sign. (best practice)

Variable Assignment | Examples



Example Creating a **price variable** in Python

```
price = 15

print(price)
```

```
15
```

We're creating a variable named **price** and assigning it **15** as an integer, then **we're using its label** (variable name) to print its value.

```
price = 15

print(price + 5)
```

```
20
```

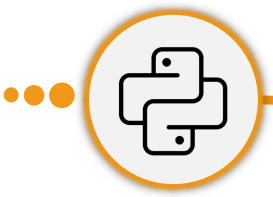
Any **operation** that can be performed **on the value** of a variable, can be performed using the **variable name**.

Here, we're adding **5** as a **hard-coded** value to the price

Hard-coded, means **not** storing into a variable

DevTip: I suggest to use variables for values **that can change** or will be **used repeatedly**.

Variable Assignment | Examples



Example Creating a **price_list variable** in Python

```
price_list = [5.5, 10.23, 7.0, 6]

type_price_list = type(price_list)

type_price_list
```

```
list
```

1. **First**, we're creating a variable named **price_list** and assigning it a list of values

2. **Then**, we're assigning the **result of type function (function call)** with our **price_list** variable as input to the variable called **type_price_list**

```
price_list = [5.5, 10.23, 7.0, 6]

print(type(price_list))
```

```
<class 'list'>
```

Note that assigning the **type( )** function to a variable here doesn't improve our program. But it's worth knowing even a function call can be assigned to a variable.

Let’s Test What We’ve learned!

- Creating Some Variables
- Assigning Values to Them
- Printing The Results

Variabe_Assignment_01.ipynb

Assignment: Variable Assignment

From: **Salar H Shamchi (izshop manager)**

Subject: **Tax Calculation**

Hi there, Welcome to the **izshop** Company!

For your first task, I want you to change the code that adds a **euro in tax** to our price before printing.

The tax will no longer be fixed at one euro, so:

- Create a "**tax**" variable and assign it a value of **1.5**
- Create a "**total**" variable equal to **"price" + "tax"**
- Print **"total"**.

Thank in advance!

Adding_Tax.ipynb

Reply Forward

----- Result Preview -----

```
price = 5
tax = 1.5
total = price + tax

print(total)
```

```
6.5
```

Solution: Variable Assignment

From: **Salar H Shamchi (izshop manager)**

Subject: **Tax Calculation**

Hi there, Welcome to the **izshop** Company!

For your first task, I want you to change the code that adds a **euro in tax** to our price before printing.

The tax will no longer be fixed at one euro, so:

- Create a "**tax**" variable and assign it a value of **1.5**
- Create a "**total**" variable equal to **"price" + "tax"**
- Print **"total"**.

Thank in advance!

Adding_Tax.ipynb

Reply Forward

----- Code Solution -----

```
price = 5
tax = 1.5
total = price + tax

print(total)
```

```
6.5
```

Overwriting Variables

You can **overwrite a variable** by assigning a new value to it.

- They can be overwritten any number of time

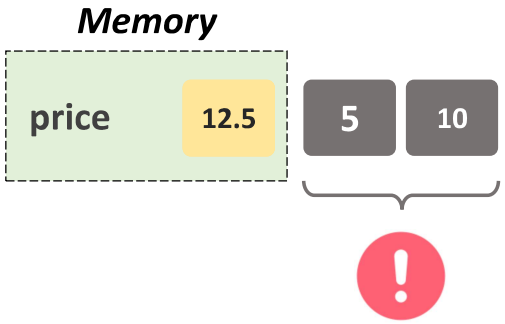
```
price = 5
price = 10
price = 12.5

print(price)
```

```
12.5
```

1. Python stores the value of 5 in **memory** when it is assigned to price.

2. Then price stores the value 10, and 5 gets removed from memory

3. Finally the price stores the value 12.5, and 10 gets removed from the memory



When you overwrite a variable, **its previous value will be lost** and **can not** be retrieved.

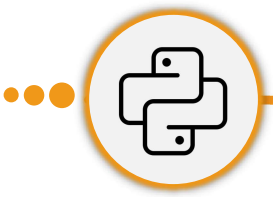
```
price = 5
new_price = 10

print(price, new_price)
```

```
5  10
```

To avoid overwriting, You need to create a new variable for a new value

Dev Tip: Use variables like **'new_something'** or **'old_something'** when testing your code to make sure you don't lose important data.

Overwriting Variables



Variables can also be **assigned as values to other variables**.

- The stored value is assigned, but there is **no connection between the variables**

```
price = 5
new_price = 10
price = new_price
new_price = 12.5

print(price, new_price)
```

```
10, 12.5
```

1. Python stores the value of `5` in memory when it is assigned to **`price`**.
2. Then assigns the value of `10` to new_price and stores it in the memory
3. Then assigns the value of new_price, which is `10`, to price and `5` is no longer assigned to any variable, so gets removed.
4. Then assigns the value of `12.5` to new_price

Memory

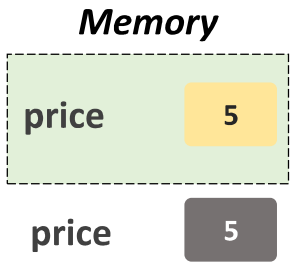
price	10	5
new_price	12.5	10

Deleting Variables

The `del` keyword will permanently **remove** variables and other objects.

```
price = 5
del price

print(price)
```

1. Python stores the value of 5 in memory when it is assigned to the variable price.
2. Then **del** keyword removes it from memory
3. When we try to print price, we get an **Error**, as the variable no longer exists in the memory.




```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 4
      1 price = 5
      2 del price
----> 4 print(price)

NameError: name 'price' is not defined
```

NameError happens when we reference a **variable** or **object** name that **doesn't exist** or isn't defined anymore!

DevTip: Deleting objects using `del` keyword generally **unnecessary**, and mostly used for large objects (like dataset with 10,000 rows) , in most cases **reassigning (overwriting)** the variables will automatically remove the old value of that object

Let’s Test What We’ve learned!

- Overwriting Variables
- Deleting Variables

Overwriting_&_Deleting_Variables_01.ipynb

Variable Naming Rules!

Variables have some basic **naming rules**

Variable names **Can**:

1. Contain letters
2. Contain numbers
3. Contain Underscores
4. Begin with a letter or underscore

Variable names **Can NOT**:

1. Start with a number
2. Contain space or other **special** characters
   (*, &, !, #, $, %, etc.)
3. Be reserved Python **keywords** i.e. for, print, max

```
help("keywords")
```

```
a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

DevTip : **snake_case** is a recommended style for naming variables in Python which is all lowercase with words separated by underscores. (my_list, price_list, first_number)

Variable Naming Rules!

Variables have some basic **naming rules**

Correct Variable Names

```
tax_rate_2023
_tax_rate_2023
TAX_RATE_2023
trnew
```

Incorrect Variable Names

```
2023_tax_rate      × starts with a number
tax_rate-2023      × has special characters
tax rate 2023      × has spaces
list               × reserved Python keyword
```

DevTip: Give your variables proper names to make your code **understandable** and more readable for **yourself** and other developers. Instead of **trnew**, consider **tax_rate_new** or **tax_new**

Tracking Variables

Use %who or %whos to track the variables you've **created** in your Jupyter notebooks

```
price = 49
product_name = "Black Boots"
date = "10-June-2023"
```

```
%who
```

```
price   product_name   date
```

%who returns variable **names**

```
%whos
```

```
Variable        Type    Data/Info
-----------------------------------
date            str     10-June-2023
price           int     49
product_name    str     Black Boots
```

%whos returns variable **names**, **types**, and **information** of defined variables

DevTip : **Magic commands** that starts with " % " **only** works in iPython environments, which applies to Jupyter Notebook and Google Colab interface.

Let’s Test What We’ve learned!

- Tracking (%who & %whos)
- Variable Naming Rules

Naming_Rules_&_Trackig_Variables_01.ipynb

Assignment: Fixing Variables

From: **Salar H Shamchi (izshop manager)**

Subject: **Price List Correction**

Hi there!

In the attached file, there is our previous intern's attempt to name a list of prices for **2023**, **2022**, **2021**. He made some errors.

Please correct the variable names that fit in the format like this: "**price_list_year**".

Thank You Buddy!

Price_Lists.ipynb

Reply | Forward

----- Result Preview -----

```
price_list_2021 = [4.5, 8.5, 17.99, 23]
price_list_2022 = [5, 7.25, 14, 25]
price_list_2023 = [6.6, 8.3, 12.5, 30]
```

```
%whos
```

```
Variable         Type    Data/Info
------------------------------
price_list_2021   list    n=4
price_list_2022   list    n=4
price_list_2023   list    n=4
```

Solution: Fixing Variables

From: **Salar H Shamchi (izshop manager)**

Subject: **Price List Correction**

Hi there!

In the attached file, there is our previous intern's attempt to name a list of prices for **2023**, **2022**, **2021**. He made some errors.

Please correct the variable names that fit in the format like this: "**price_list_year**".

Thank You Buddy!

Price_Lists.ipynb

Reply | Forward

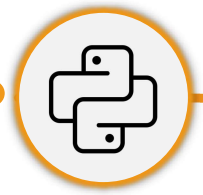
----- Code Solution -----

```
price_list_2021 = [4.5, 8.5, 17.99, 23]
price_list_2022 = [5, 7.23, 14, 25]
price_list_2023 = [6.6, 8.3, 12.5, 30]
```

```
%whos
```

```
Variable          Type    Data/Info
------------------------------
price_list_2021   list    n=4
price_list_2022   list    n=4
price_list_2023   list    n=4
```

Section Wrap-Up



✔ Variables are containers used to **store values** from any data type

- These values are stored in memory and can be referenced repeatedly in your code

✔ **Overwrite (Reassign) variables** by assigning new values to them

- The **old** value will be **lost** unless it's assigned to another variable (**old_something, new_something**)

✔ Variables names must follow **Python's naming rules**

- "**snake_case**" is the recommended naming style (all lowercase with underscores separating each word)

✔ Give your variables **intuitive** names

- Although we can use **magic commands (%)** to track variables, a good name can save a lot of times