

# Python review

# Overview

Jupyter notebook on Ed Lesson

functions and variables

control


data structures

numpy/pandas

# Jupyter notebook on Ed Lesson

side bar

notebook cells (code, text)

run (  or `shift+enter` or `ctrl+enter` )

autocomplete / syntax highlighting

---

markdown syntax: <https://www.markdownguide.org/basic-syntax/>

# Guide to Using Lab Notebook

in-class exercises

notes

study guide

to reset: ... > Reset to Scaffold

# Functions and Variables

# Hello to You

```
What's your name? John  
Hello, John!
```

# Anatomy

```
answer = input("What's your name? ")

print("Hello answer")           # wrong
print("Hello", answer)         # multiple arguments
print(f"Hello {answer}")       # f-string
```

- **Function:** `input()`
- **Argument:** "What's your name? "
- **Side effect:** prompt the user and wait for input
- **Return values:** user input
- **Variable:** `answer`

# Hello Function

```
hello()  
# Output: Hello, World!
```

```
hello("John")  
# Output: Hello, John
```



# def

```
def hello():  
    print("Hello world")  
  
answer = input("What's your name? ")  
  
hello()
```

# Arguments

```
def hello(to):  
    print("Hello ", to)  
  
answer = input("What's your name? ")  
hello(answer)
```

# Arguments with default values

```
def hello(to="world"):
    print("Hello ", to)

answer = input("What's your name? ")

hello(answer)
# Output: Hello {answer}

hello()
# Output: Hello world
```

# Positional vs. keyword arguments

```
# positional arguments  
hello(answer)
```

```
# keyword arguments  
hello(to=answer)
```

# Arguments 🤔

```
def hello(a, b="Doe"):
    print("Hello", a, "and", b)

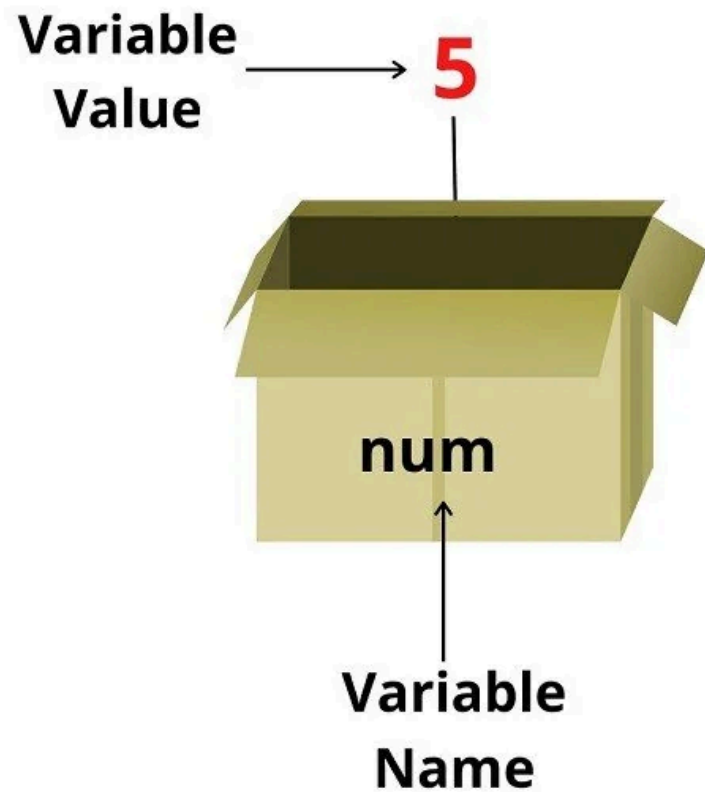
hello("John", "Doe")
hello("Doe", "John")
hello(b="Doe", a="John")
hello("John")
hello(b="John")
```

# Scope

```
def main():  
    answer = input("What's your name? ")  
    hello()  
  
def hello():  
    print("Hello ", answer)  
  
main()
```

# Passing variables to functions

```
def main():  
    answer = input("What's your name? ")  
    hello(answer)  
  
def hello(to):  
    print("Hello ", to)  
  
main()
```





# Scope 🤔

```
def func1():  
    x = 10  
    y = 20  
    print(x + y)
```

```
def func2(y):  
    x = 30  
    print(x + y)
```

```
func1()
```

```
func2(2)
```

```
x = 20  
func2(x)
```

# return

```
def main():  
    answer = input("What's your name? ")  
    message = hello_message(answer)  
  
    print(message)  
  
def hello_message(to="world"):  
    msg = "Hello " + to  
  
    return msg  
  
main()
```



# 1. Area calculator

1. Define a function `calculate_area(length, width)` that:
  - Takes two parameters: `length` and `width`.
  - Returns the area of the rectangle ( $\text{area} = \text{length} \times \text{width}$ ).
2. Ask the user to input the length and width of the rectangle.
3. Call the function with the user-provided values and print the area and perimeter.

## Expected output:

```
Enter the length of the rectangle: 5
Enter the width of the rectangle: 3
The area of the rectangle is 15
```



# 1. Area calculator (solution)

```
# Main program
def main():
    # Ask the user for the length and width of the rectangle. Use input() and float().
    length = float(input("Enter the length of the rectangle: "))
    width = float(input("Enter the width of the rectangle: "))

    # Call the functions to calculate area and perimeter
    area = calculate_area(length, width)

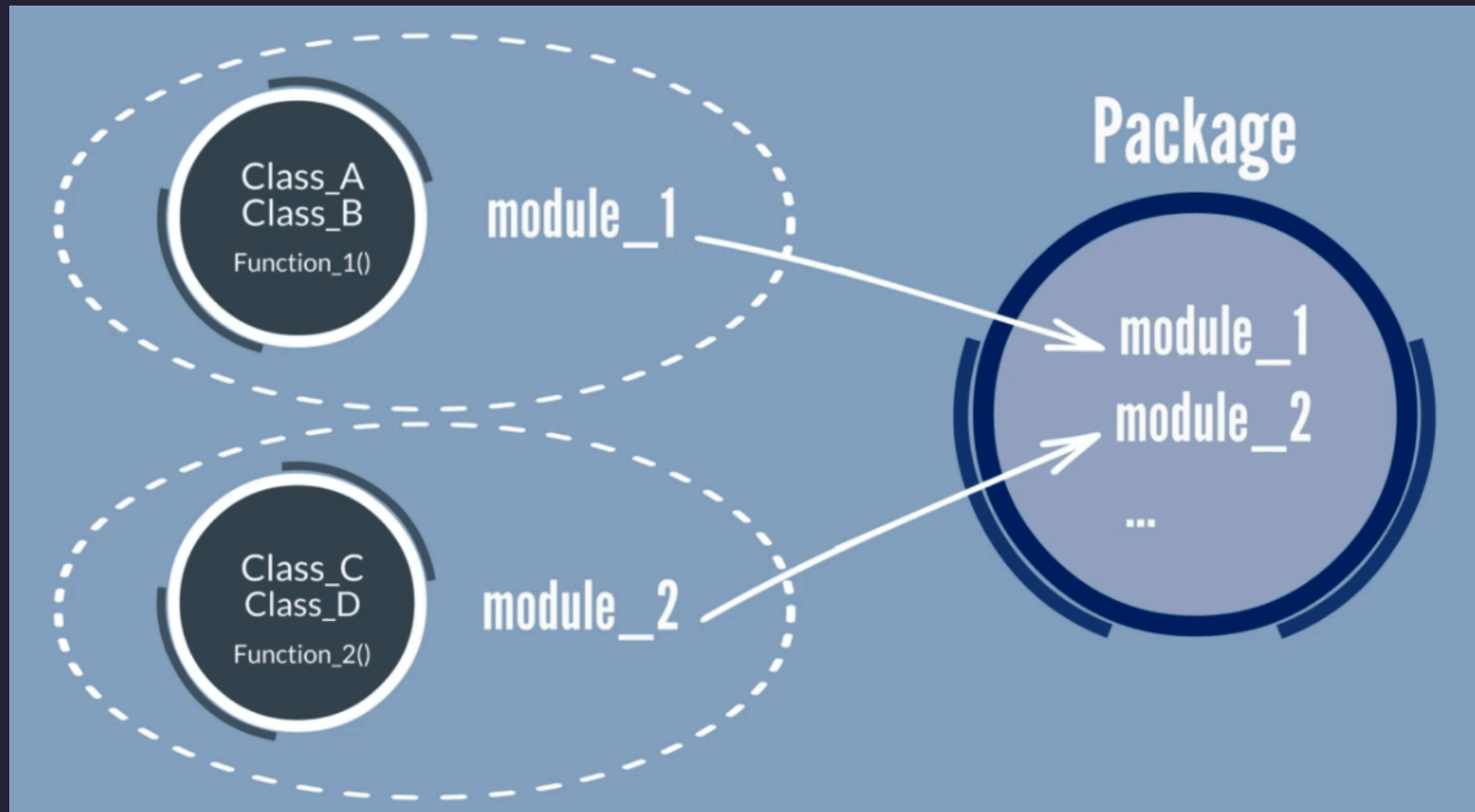
    # Print the results
    print(f"The area of the rectangle is: {area}")

# Function to calculate the area of a rectangle
def calculate_area(length, width):
    area = length * width
    return area

# Call the main function to run the program
main()
```

# Packages

**functions < modules < packages = libraries**



# `import` module

```
# math.py
import math

print(math.pi)
print(math.sqrt(4))
print(math.pow(2, 3))
print(math.floor(3.14))
print(math.ceil(3.14))
print(math.factorial(5))
```

**from** **module** **import** functions, variables, etc.

```
from math import pi, sqrt  
  
print(pi)  
print(sqrt(4))
```

**as** **to give alias**

```
import math as m  
  
print(m.pi)  
print(m.sqrt(4))
```



# Import custom functions

```
# ./my_module.py
def hello(to="world"):
    return f"Hello {to}"
```

```
# ./main.py
from my_module import hello

hello("John")
```

# Control

- **conditionals:** branching
- loops: repetition



## Is x less than y?

```
Enter a number: 5  
Enter another number: 3  
x is greater than y
```

```
Enter a number: 2  
Enter another number: 2  
x is equal to y
```

```
Enter a number: 1  
Enter another number: 4  
x is less than y
```

# Comparison operators

>

<

>=

<=

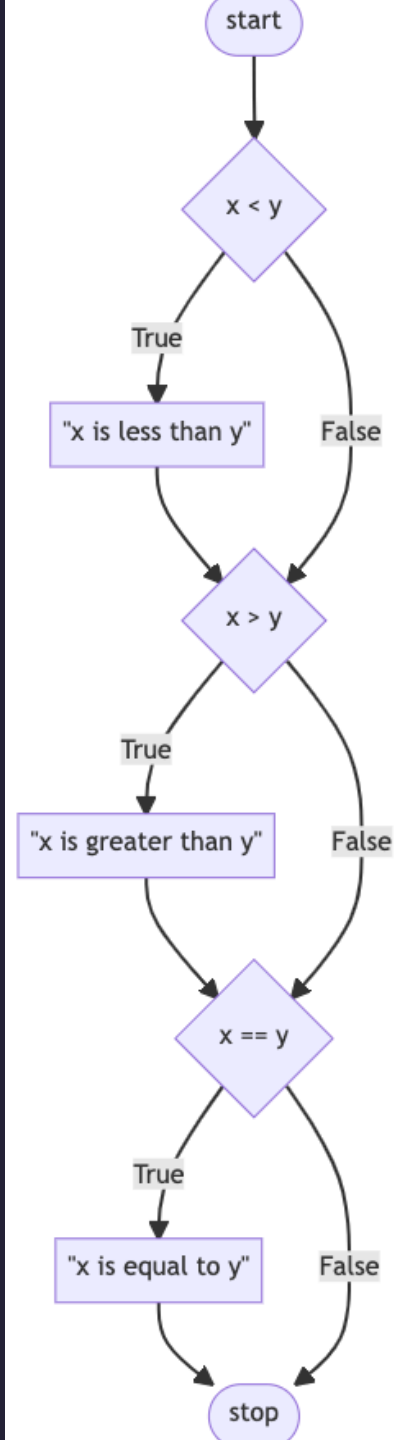
==

!=

# if

```
x = int(input("Enter a number: "))
y = int(input("Enter another number: "))

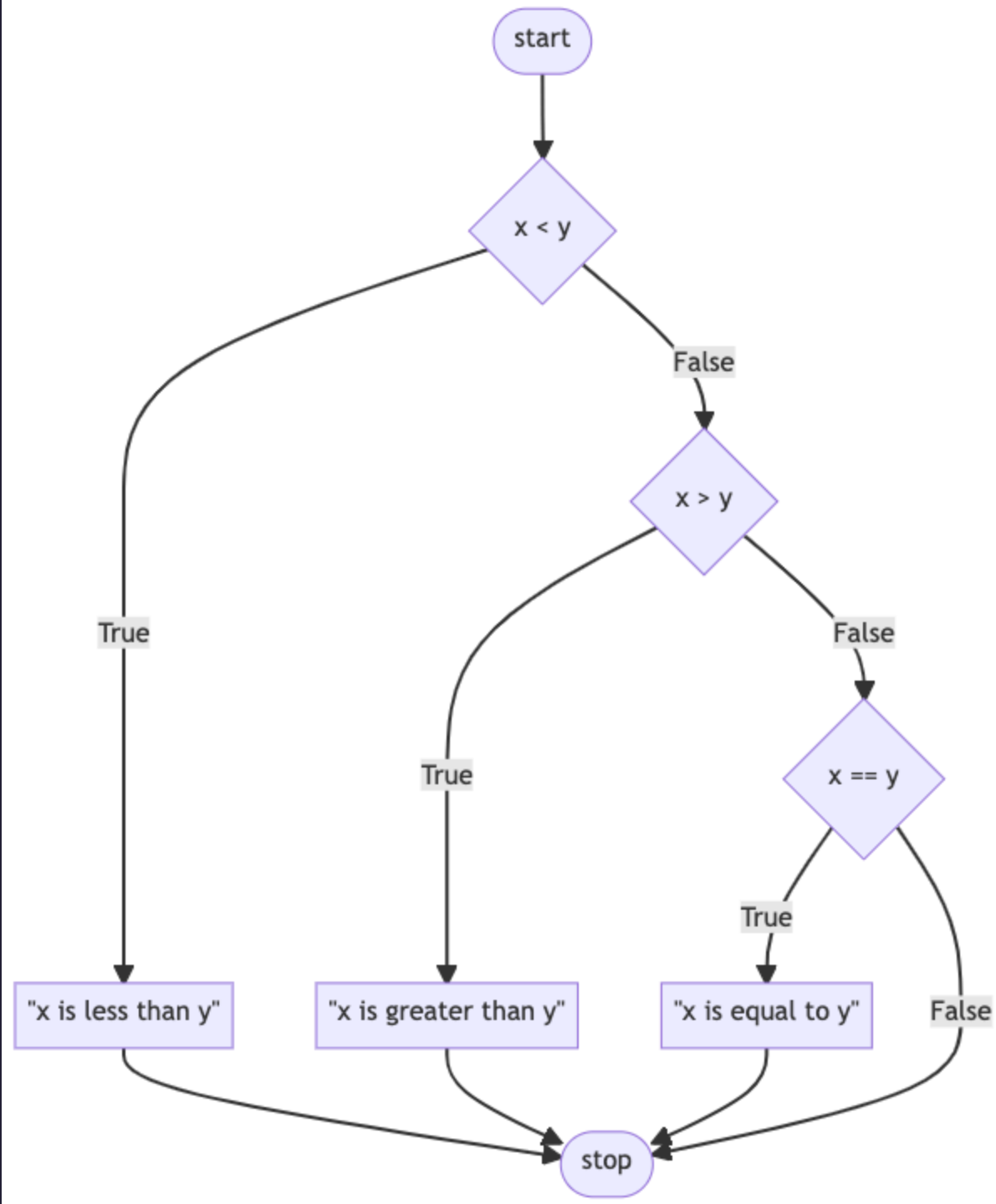
if x < y:
    print("x is less than y")
if x > y:
    print("x is greater than y")
if x == y:
    print("x is equal to y")
```



## elif (else if)

```
x = int(input("Enter a number: "))
y = int(input("Enter another number: "))

if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
elif x == y:
    print("x is equal to y")
```

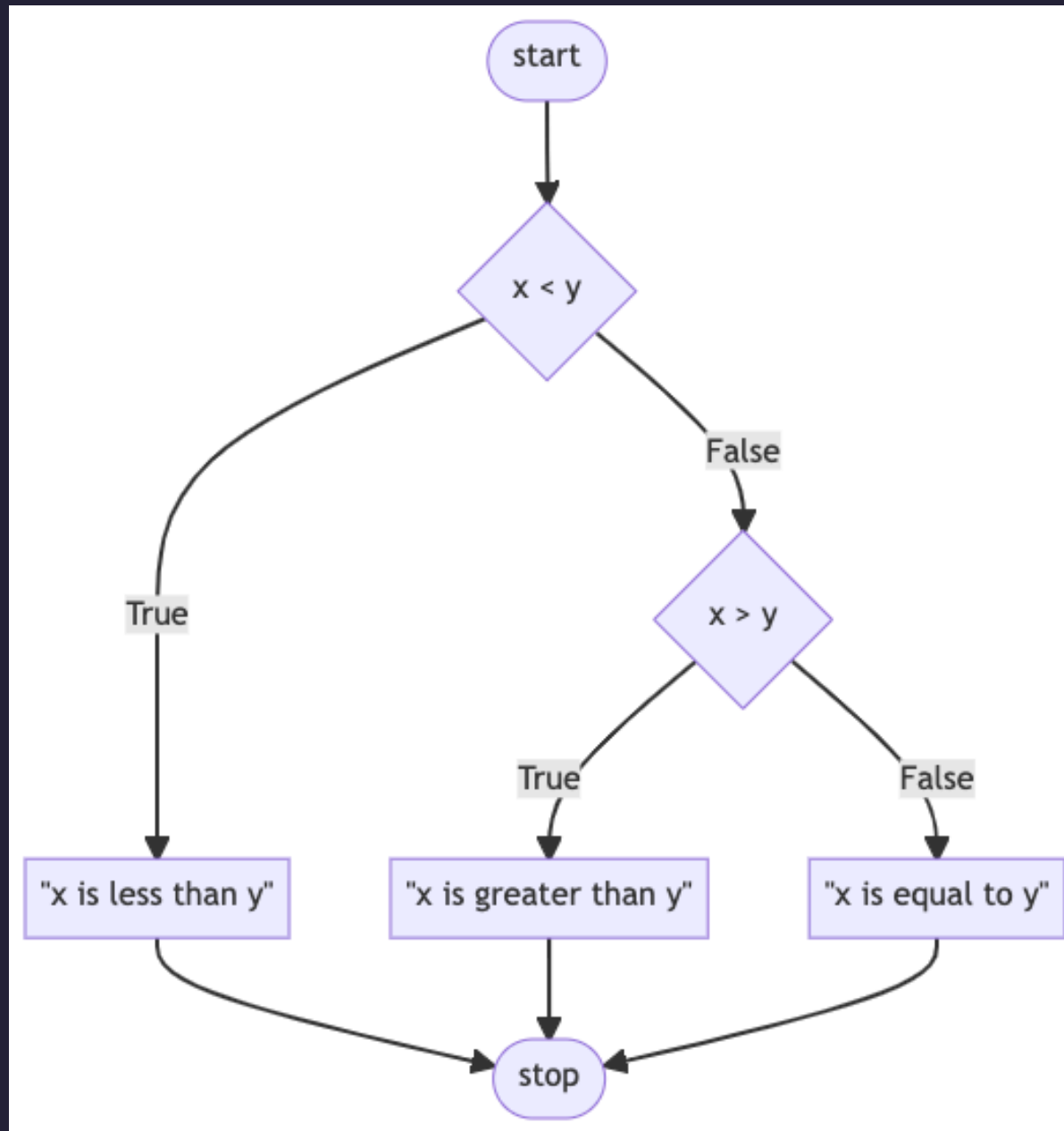




## else

```
x = int(input("Enter a number: "))
y = int(input("Enter another number: "))

if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x is equal to y")
```



# Logical operators

- `or`
- `and`
- `not`

---

<https://www.datacamp.com/tutorial/python-logical-operators-introduction>

or

```
x = int(input("Enter a number: "))
y = int(input("Enter another number: "))

if x < y or x > y:
    print("x is not equal to y")
else:
    print("x is equal to y")
```

# What am I not A? 🤔

```
score = int(input("Enter your score: "))

if score >= 85:
    print("A")
if score >= 80:
    print("A-")
if score >= 75:
    print("B+")
if score >= 70:
    print("B")
if score >= 65:
    print("B-")
```

# Conditions = **bool** expressions (True or False)

```
print(2 > 1) # True

print(5 % 2 == 0) # False

if x % 2 == 0: # True or False
    print("x is even")

if x > y: # True or False
    print("x is greater than y")

if score >= 85: # True or False
    print("A")

if True:
    print("always get printed")

if False:
    print("never get printed")
```

## A or B?

```
if 2 > 1:  
    print("A")  
  
else:  
    print("B")
```

## A or B?

```
if 2 == 1:  
    print("A")  
  
else:  
    print("B")
```



## A or B?

```
if 2 > 1 or 2 == 1:  
    print("A")  
  
else:  
    print("B")
```

# bool: True or False

```
def main():
    x = int(input("Enter a number: "))

    x_is_even = is_even(x)

    if x_is_even:    # True or False from is_even()
        print("x is even")
    else:
        print("x is odd")

def is_even(x):
    if x % 2 == 0:
        return True
    else:
        return False

main()
```

# Control

- conditionals: branching
- **loops**: repetition



# Don't Repeat Yourself (DRY)

```
print("meow")  
print("meow")  
print("meow")
```

# Loop

`while`

`for`

## **while**: conditionally repeated

```
i = 3
while i > 0:
    print("meow")
    i = i - 1
# or
# i -= 1
```

# Assignment operators

- `=`
- `+=`
- `-=`
- `*=`
- `/=`
- ...

---

<https://python-reference.readthedocs.io/en/latest/docs/operators/>

## **for**: repeat over a sequence (list, string, ...)

```
for i in [0, 1, 2]:  
    print("meow")
```

```
for i in [0, 0, 0]:  
    print("meow")
```

```
for i in "abc":  
    print("meow")
```



## range()

```
for i in range(3):  
    print("meow")  
  
for i in range(0, 3):    # range(3)  
    print(i)  
  
for i in range(5, 9):  
    print(i)
```



## 2. Printing even numbers between 1 and 20

- Use a `for` loop and the `range` function to iterate from 1 to 20 (inclusive).
- Inside the loop, use an `if` statement to check if the current number is even.
  - To check for evenness, use the modulo operator `%`.
- If the number is even, print it.

```
2
4
6
8
10
12
14
16
18
20
```

## 2. Printing even numbers between 1 and 20 (solution)

```
for i in range(1, 21):  
    if i % 2 == 0:  
        print(i)
```

# Interactive meow

```
Enter a positive number: -3
Enter a positive number: -1
Enter a positive number: 4
meow
meow
meow
meow
```

# Infinite loop

```
while True:  
    print("meow")
```

## How to get out of a loop? **continue**, **break**

```
while True:          # infinite loop
    n = int(input("Enter a positive number: "))

    if n < 0:         # if n is negative
        continue    # continue to next iteration
    else:             # if n is positive
        break        # break out of the loop

for _ in range(n):
    print("meow")
```



## infinite loop **continue**s anyway

```
while True: # infinite loop
    n = int(input("Enter a positive number: "))

    if n > 0: # if n is positive
        break # break out of the loop

for _ in range(n):
    print("meow")
```



## return to break out of a loop

```
def main():  
    n = get_positive_number()  
    meow(n)  
  
def get_positive_number():  
    while True:  
        n = int(input("Enter a positive number: "))  
        if n > 0:  
            return n # return the number  
  
def meow(n):  
    for _ in range(n):  
        print("meow")  
  
main()
```



## 3. Input Validation for Even Numbers

- Use a `while True` loop to prompt the user to enter a number until an even number is entered.
- Write a function `is_even(n)` that takes an integer `n` and returns `True` if `n` is even and `False` otherwise.
- Use an `if` statement to check whether the entered number is even.
  - If the number is even, return it and break out of the loop.
- Calculate the square of the returned even number and print it.

```
Enter an even number: 3
3 is not an even number. Try again.
Enter an even number: 5
5 is not an even number. Try again.
Enter an even number: 8
64
```



## 3. Input Validation for Even Numbers (solution)

```
def main():
    number = get_even_number()
    print(number ** 2)

def get_even_number():
    while True:
        n = int(input("Enter an even number: "))
        if is_even(n):
            return n
        print(f"{n} is not an even number. Try again.")

def is_even(n):
    return n % 2 == 0

main()
```

# Data structures

list

dict

## **list**: a list of (any) values

```
cities = ["Montreal", "Toronto", "Vancouver", "Detroit"]  
numbers = [1, 2, 3, 4, 5]  
any_type_you_want = [1, "meow", 3.14, True]
```

---

<https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>

# Access item using index

```
# Access
cities = ["Montreal", "Toronto", "Vancouver", "Detroit"]

print(cities)      # ["Montreal", "Toronto", "Vancouver", "Detroit"]
print(cities[0])   # Montreal
print(cities[1])   # Toronto
print(cities[2])   # Vancouver
print(cities[3])   # Detroit
print(cities[4])   # ???

# Update
cities[0] = "New York"

print(cities)      # ["New York", "Toronto", "Vancouver", "Detroit"]
```

## Add item to list - **append**

```
cities = ["Montreal", "Toronto", "Vancouver", "Detroit"]
cities.append("New York")

print(cities) # ["Montreal", "Toronto", "Vancouver", "Detroit", "New York"]
```

## Delete item from list - **del**

```
cities = ["Montreal", "Toronto", "Vancouver", "Detroit"]

del cities[0]
print(cities) # ["Toronto", "Vancouver", "Detroit"]

del cities[0]
print(cities) # ["Vancouver", "Detroit"]
```

# Search: Is this **in** the list or **not in** the list?

## Membership operators

```
cities = ["Montreal", "Toronto", "Vancouver", "Detroit"]

if "Montreal" in cities:
    print("Montreal is in the list")

if "New York" not in cities:
    print("New York is not in the list")
```



`len()`, `min()`, `max()`, `sum()`

```
numbers = [1, 2, 3, 4, 5]

print(len(numbers))      # 5
print(min(numbers))      # 1
print(max(numbers))      # 5
print(sum(numbers))      # 15
```

---

<https://docs.python.org/3/library/functions.html>

# Loop over list

1. `for`

```
cities = ["Montreal", "Toronto", "Vancouver", "Detroit"]  
  
for city in cities:  
    print(city)
```

2. `len()` & `range()`

```
cities = ["Montreal", "Toronto", "Vancouver", "Detroit"]  
length = len(cities)  
  
for i in range(length):  
    print(cities[i])
```

# **dict**ionary: a collection of key-value pairs

```
cities = {  
    "key": "value",  
    "name": "Montreal",  
    "state": "QC",  
    "country": "CA"  
}
```

---

<https://docs.python.org/3/tutorial/datastructures.html#dictionaries>

# Access item in dict using **key**

```
cities = {  
    "name": "Montreal",  
    "state": "QC",  
    "country": "CA"  
}  
  
print(cities["name"])      # Montreal  
print(cities["state"])     # QC  
print(cities["country"])   # CA  
  
# Update  
cities["name"] = "New York"  
  
# Add  
cities["continent"] = "NA"
```

## Delete item from dict - `del`

```
cities = {  
    "name": "Montreal",  
    "state": "QC",  
    "country": "CA"  
}  
  
del cities["state"]  
  
print(cities)
```

# `keys()`, `values()`, `items()` return list-like objects

```
cities = {
    "name": "Montreal",
    "state": "QC",
    "country": "CA"
}

print(cities.keys())
# output: dict_keys(['name', 'state', 'country'])

print(cities.values())
# output: dict_values(['Montreal', 'QC', 'CA'])

print(cities.items())
# output: dict_items([('name', 'Montreal'), ('state', 'QC'), ('country', 'CA')])
```

## Search: **in**, **not in**

```
cities = {  
    "name": "Montreal",  
    "state": "QC",  
    "country": "CA"  
}  
  
if "name" in cities.keys():  
    print("name is in the keys of the dict")  
  
if "Detroit" not in cities.values():  
    print("Detroit is not in the values of the dict")
```

`len()`, `min()`, `max()`, `sum()`

```
numbers = {  
    "a": 1,  
    "b": 2,  
    "c": 3,  
    "d": 4,  
    "e": 5  
}  
  
print(len(numbers))      # 5  
print(min(numbers))     # a (key)  
print(max(numbers.values())) # 5 (value)  
print(sum(numbers.values())) # 15
```



# loop over dict

```
cities = {  
    "name": "Montreal",  
    "state": "QC",  
    "country": "CA"  
}  
  
for key in cities: # equivalent to cities.keys()  
    print(key)  
  
for key in cities: # equivalent to cities.keys()  
    print(key, cities[key])  
  
for value in cities.values():  
    print(value)  
  
for key, value in cities.items():  
    print(key, value)
```

# Accessing items in a list/tuple/dict

- list/tuple: `[index]`
- dict: `[key]`

# How to access the following items?

```
cities = [  
    {"name": "Montreal", "state": "QC", "country": "CA"},  
    {"name": "Toronto", "state": "ON", "country": "CA"},  
    {"name": "Vancouver", "state": "BC", "country": "CA"},  
    {"name": "Detroit", "state": "MI", "country": "US"}  
]
```

- {"name": "Vancouver", "state": "BC", "country": "CA"}
- "Vancouver"
- "Montreal"

```
cities = {  
    "name": ["Montreal", "Toronto", "Vancouver", "Detroit"],  
    "state": ["QC", "ON", "BC", "MI"],  
    "country": ["CA", "CA", "CA", "US"]  
}
```

- ["Montreal", "Toronto", "Vancouver", "Detroit"]
- "Montreal"

```
cities = {
    "location": {
        "Montreal": {"state": "QC", "country": "CA"},
        "Toronto": {"state": "ON", "country": "CA"},
        ...
    },
    "stats": {
        "Montreal": [
            {"year": 2013, "population": 2000000, "area": 431.5},
            {"year": 2014, "population": 1980000, "area": 431.5}
        ],
        "Toronto": [
            {"year": 2013, "population": 2800000, "area": 630.2},
            ...
        ],
        ...
    }
}
```

- {"year": 2013, "population": 2000000, "area": 431.5}
- "QC"
- 2000000

	List	Dict
Access	<code>cities[0]</code>	<code>cities["key"]</code>
Update	<code>cities[0] = "new item"</code>	<code>cities["existing key"] = "new value"</code>
Add	<code>cities.append("new item")</code>	<code>cities["new key"] = "new value"</code>
Delete	<code>del cities[0]</code>	<code>del cities["key"]</code>

	List	Dict
Join	<code>cities3 = cities1 + cities2</code>	<code>cities3 = cities1   cities2</code>
Search	<code>"item" in cities</code>	<code>"key" in cities.keys()</code> <code>"value" in cities.values()</code>
Loop	<code>for item in cities:</code>	<code>for key in cities.keys():</code> <code>for value in cities.values():</code> <code>for item in cities.items():</code>
Sort	<code>cities.sort()</code>	<code>sorted(cities)</code>



## 4. Citybook

- Update the dictionary to include the population and area for Montreal.
  - population: 1704694, area: 431.5
- Write a function `calc_density` that takes a city's population and area as input and returns the population density.
- Update the dictionary to include the population density for Montreal.
- Use a `for` loop to iterate over the dictionary.
  - Print out the city's name, its state/province, country, population, and area.
- Finally, print the total number of fields stored in the dictionary.

```
name: Montreal
state: QC
country: CA
population: 1704694
area: 431.5
density: 3952.0
Total number of fields: 6
```





## 4. Citybook (solution)

```
def calc_density(pop, area):  
    return pop/area  
  
cities = {  
    "name": "Montreal",  
    "state": "QC",  
    "country": "CA"  
}  
  
pop = 1704694  
area = 431.5  
density = calc_density(pop, area)  
  
cities["population"] = pop  
cities["area"] = area  
cities["density"] = density  
  
for key, value in cities.items():  
    print(key + ": " + str(value))  
  
print("Total number of keys:", len(cities))
```

# Vectorization using Numpy and Pandas

# list and dict are not ideal for data analysis

```
# height and weight
data = [
    [170, 68],
    [180, 70],
    [160, 60],
    [150, 55],
    [175, 65]
]

# average height
total_height = 0
for row in data:
    total_height += row[0]
average_height = total_height / len(data)
print(average_height)

# bmi
for row in data:
    height = row[0]
    weight = row[1]
    bmi = weight / (height / 100) ** 2
    print(bmi)
```

# Vectorization

```
import numpy as np

data = np.array([
    [170, 68],
    [180, 70],
    [160, 60],
    [150, 55],
    [175, 65]
])

# select height and weight
height = data[:, 0]
weight = data[:, 1]

# average height
average_height = np.mean(height)

# bmi
bmi = weight / (height / 100) ** 2
```

# Numpy array for matrix

$$np_{1d} = [1 \quad 2 \quad 3 \quad 4 \quad 5]$$

$$np_{2d} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
np_1d = np.array([1, 2, 3, 4, 5])  
print(np_1d.shape) # (5,)
```

```
np_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
print(np_2d.shape) # (3, 3)
```

# Subsetting

$$np_{2d} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
np_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
print(np_2d[0])      # [1 2 3]
print(np_2d[0, 1])   # 2
print(np_2d[:, 1])    # [2 5 8]
print(np_2d[0, :])    # [1 2 3]
print(np_2d[1:, :])   # [[4 5 6] [7 8 9]]
```

# Filtering

```
np_1d = np.array([1, 2, 3, 4, 5])  
  
cond = np_1d > 3    # [False False False  True  True]  
print(np_1d[cond])  # [4 5]
```

# Mathematical operations

```
np_1da = np.array([1, 2, 3, 4, 5])
np_1db = np.array([6, 7, 8, 9, 10])

print(np_1da + np_1db) # [ 7  9 11 13 15]
print(np_1da - np_1db) # [-5 -5 -5 -5 -5]
print(np_1da * np_1db) # [ 6 14 24 36 50]

print(np.mean(np_1da)) # 3.0
print(np.sum(np_1da))   # 15
print(np.std(np_1da))   # 1.4142135623730951
```



# Pandas Series and DataFrame

Series			Series			DataFrame		
	apples			oranges			apples	oranges
0	3	+	0	0	=	0	3	0
1	2		1	3		1	2	3
2	0		2	7		2	0	7
3	1		3	2		3	1	2

# Pandas DataFrame for tabular data

```
import pandas as pd

data = {
    'name': ['John', 'Jane', 'Mary'],
    'age': [25, 30, 27]
}
df = pd.DataFrame(data)

print(df.index)
print(df.columns)
print(df.head())
```

# Subsetting

## Selecting columns

```
df['name']  
df[['name', 'age']]
```

## Selecting rows

```
df.loc[0]  
df.loc[0:2]
```

## Selecting rows and columns

```
df.loc[0, 'name']  
df.loc[0:2, ['name', 'age']]
```

# Filtering

```
cond = df['age'] > 25  
df[cond]  
df.loc[cond]  
cond2 = (df['age'] > 25) & (df['name'] == 'John')  
df.loc[cond2, 'name']
```

# Mathematical operations

```
df['age'] + 5  
df['age'] * 2
```

```
df['age'].mean()  
df['age'].sum()
```

```
df['bmi'] = df['weight'] / (df['height'] / 100) ** 2
```



## 5. HR Data Analysis (pandas or numpy)

1. Create a Pandas DataFrame (or Numpy array) from the employee data.
2. Use filtering to select employees from the "IT" department.
3. Use another filter to select employees with a salary greater than \$60,000.
4. Calculate the average salary of all employees.
5. Calculate the average salary of the employees in the "IT" department.

Name	Age	Department	Salary
John	28	IT	55000
Alice	34	HR	62000

```
import pandas as pd

# Step 1: Create a DataFrame with employee data
data = {
    'Name': ['John', 'Alice', 'Bob', 'Claire', 'Dan', 'Eva'],
    'Age': [28, 34, 45, 29, 40, 32],
    'Department': ['IT', 'HR', 'IT', 'Finance', 'IT', 'HR'],
    'Salary': [55000, 62000, 70000, 48000, 72000, 59000]
}

df = pd.DataFrame(data)

# Step 2: Subset employees from the IT department
it_department = df[df['Department'] == 'IT']
print("Employees in the IT department:")
print(it_department)

# Step 3: Filter employees with a salary greater than 60,000
high_salary = df[df['Salary'] > 60000]
print("\nEmployees with salary greater than 60,000:")
print(high_salary)

# Step 4: Calculate the average salary of all employees
average_salary = df['Salary'].mean()
print(f"\nThe average salary of all employees is: {average_salary}")

# Step 5: Calculate the average salary of IT employees
average_salary_it = it_department['Salary'].mean()
print(f"The average salary of IT employees is: {average_salary_it}")
```

```

import numpy as np

# Step 1: Create a 2D NumPy array with employee data
# Employee data: [Name, Age, Department, Salary]
employee_data = np.array([
    ['John', 28, 'IT', 55000],
    ['Alice', 34, 'HR', 62000],
    ['Bob', 45, 'IT', 70000],
    ['Claire', 29, 'Finance', 48000],
    ['Dan', 40, 'IT', 72000],
    ['Eva', 32, 'HR', 59000]
], dtype=object) # Use object dtype to hold both strings and numbers

# Step 2: Subset employees from the IT department using NumPy
it_department_mask = employee_data[:, 2] == 'IT' # Mask for IT department
it_employees = employee_data[it_department_mask]
print("Employees in the IT department:")
print(it_employees)

# Step 3: Filter employees with a salary greater than 60,000 using NumPy
high_salary_mask = employee_data[:, 3] > 60000 # Mask for salary > 60,000
high_salary_employees = employee_data[high_salary_mask]
print("\nEmployees with salary greater than 60,000:")
print(high_salary_employees)

# Step 4: Calculate the average salary of all employees using NumPy
average_salary = np.mean(salaries)
print(f"\nThe average salary of all employees is: {average_salary}")

# Step 5: Calculate the average salary of IT employees using NumPy
average_salary_it = np.mean(it_employees[:, 3])
print(f"The average salary of IT employees is: {average_salary_it}")

```