Control

Recap

- `if` / `elif` / `else`
- comparison operators
- logical operators
- boolean values & expressions
- control flow usign `return`

⏪ Recap

```python
if x < 10:
    print('A')
elif x >= 13:
    print('B')
elif x >= 20:
    print('C')
else:
    print('D')
```

Control

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

1. Meow

Don't Repeat Yourself (DRY)

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

`while`

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

`while` using `-=` for assignment

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

Assignment operators

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

...

---

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

`for`

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

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

`range()`

```python
for i in range(3):
    print("meow")
```

_ for throwaway variable

```
for _ in range(3):
    print("meow")
```

`range(start, end)`

```python
for i in range(0, 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
```

3. Interactive meow

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

Infinite loop

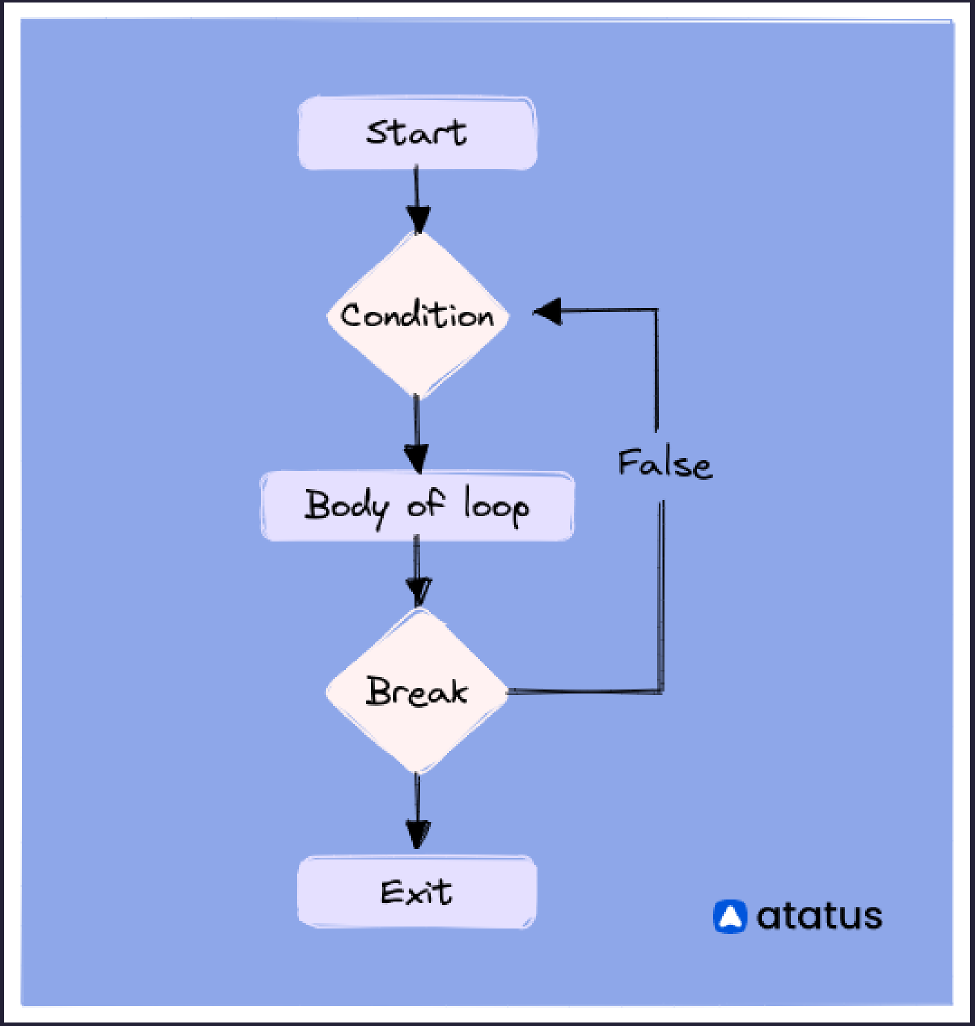
```python
while True:
    print("meow")
```

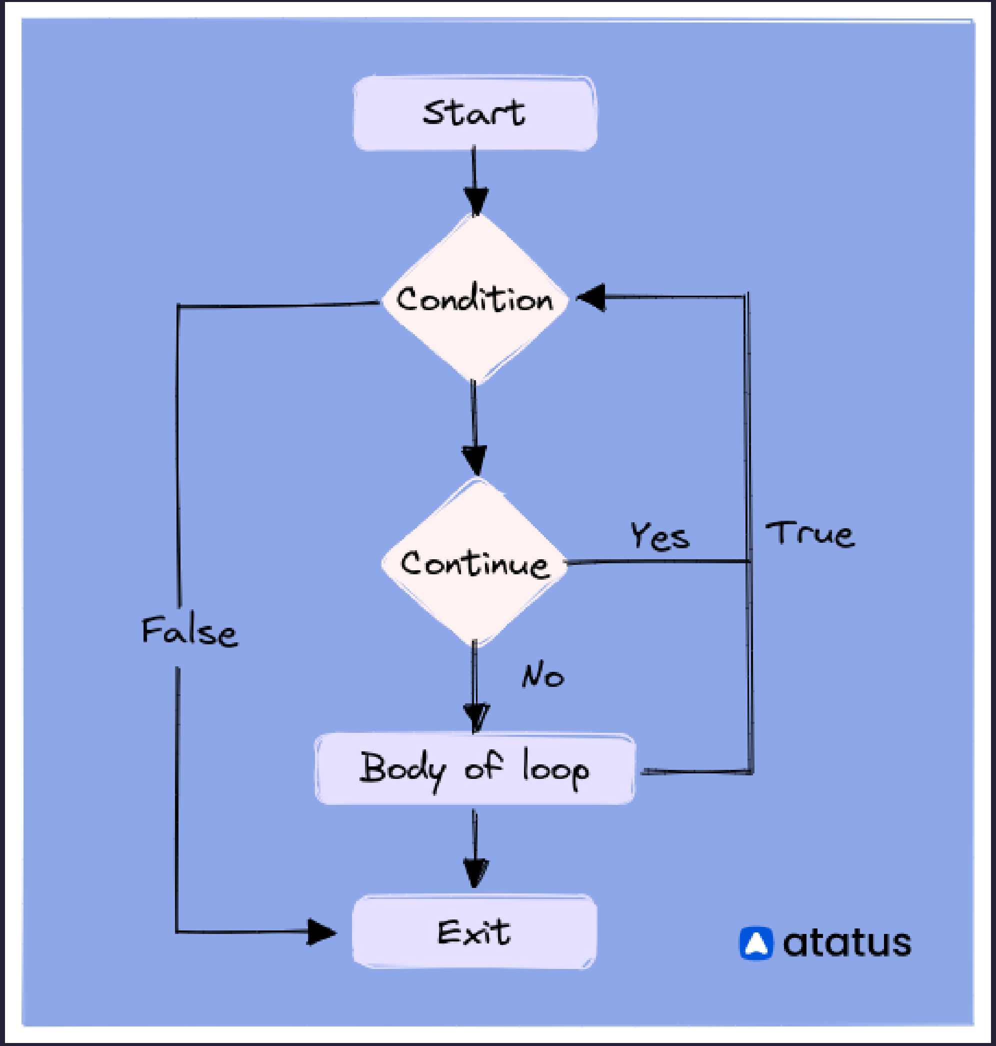
loop control using `continue`, `break`

```python
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")
```


Start
Condition
Body of loop
False
Break
Exit
atatus


Start
Condition
Continue
Yes
True
False
No
Body of loop
Exit
atatus

infinite loop continues 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

```python
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()
```

💻 4. 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
```

List

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

- access
- add
- delete
- update
- sort
- loop

---

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

Access list item

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

print(cities[0]) # Montreal
print(cities[1]) # Toronto
print(cities[2]) # Vancouver
print(cities[3]) # Detroit
```

Add item to list - `append`

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

cities.append("New York")

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

Delete item from list - `del`

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

del cities[0]

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

del cities[0]

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

Update item in list

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

cities[0] = "New York"

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

Join two lists - `+`, `extend`

```python
cities = ["Montreal", "Toronto", "Vancouver", "Detroit"]
cities2 = ["New York", "Boston", "Chicago"]

cities3 = cities + cities2
print(cities3)

cities.extend(cities2)
print(cities)
```

Sort list - `sort`

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

cities.sort()
print(cities)

cities.sort(reverse=True)
print(cities)
```

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

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

print(len(numbers))
print(min(numbers))
print(max(numbers))
print(sum(numbers))
```

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

Loop over list - `for`

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

for city in cities:
    print(city)
```

Loop over list - `len()` & `range()`

```
cities = ["Montreal", "Toronto", "Vancouver", "Detroit"]
length = len(cities)

for i in range(length):
    print(cities[i])
```

Membership opeartors: `in`, `not in`

```python
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")
```

`dict`ionary

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

- access
- add
- delete
- update
- sort
- loop

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

`keys()`, `values()`, `items()`

```
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')])
```

Access dict item

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

print(cities["name"])
print(cities["state"])
print(cities["country"])
```

Add item to dict

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

cities["continent"] = "NA"

print(cities)
```

Delete item from dict - `del`

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

del cities["state"]

print(cities)
```

Update item in dict

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

cities["name"] = "New York"

print(cities)
```

Join two dicts (unique keys) - `|`, `update`

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

cities2 = {
    "name2": "New York",
    "state2": "NY",
    "country2": "US"
}

cities3 = cities | cities2
print(cities3)

cities.update(cities2)
print(cities)
```

Sort dict - `sorted`

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

sorted(cities.items())
```

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)
```

💻 6. 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
```

More on data structures

- list of dict: `[{...}, {...}, {...}]`
- list of list: `[[...], [...], [...]]`
- dict of list: `{"key1": [...], "key2": [...], "key3": [...]}`
- dict of dict: `{"key1": {...}, "key2": {...}, "key3": {...}}`

list of dict

```python
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"}
]

print(type(cities))

for city in cities:
    print(type(city))
    print(city["name"], city["state"], city["country"])

# change the state of Montreal to NY
cities[0]["state"] = "NY"
print(cities[0]["state"])
```

list of list

```python
cities = [
    ["Montreal", "QC", "CA"],
    ["Toronto", "ON", "CA"],
    ["Vancouver", "BC", "CA"],
    ["Detroit", "MI", "US"]
]
print(type(cities))

for city in cities:
    print(type(city))
    print(city[0], city[1], city[2])

# change the state of Montreal to NY
cities[0][1] = "NY"
print(cities[0][1])
```

dict of list

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

for col, rows in cities.items():
    print(type(rows))

    for row in rows:
        print(col + ": " + row)

# change the state of Montreal to NY
cities["state"][0] = "NY"
print(cities["state"][0])
```

dict of dict

```
cities = {
    "Montreal": {"state": "QC", "country": "CA"},
    "Toronto": {"state": "ON", "country": "CA"},
    "Vancouver": {"state": "BC", "country": "CA"},
    "Detroit": {"state": "MI", "country": "US"}
}
print(type(cities))

for city, info in cities.items():
    print(type(info))
    print(city, info["state"], info["country"])

# change the state of Montreal to NY
cities["Montreal"]["state"] = "NY"
print(cities["Montreal"]["state"])
```

💻👪 Citybook2

1. Write pseudocode in `main()`
2. Draft the definition of each function
3. Complete the function definitions.
4. Complete `main()` with function calls

```
Enter a city name: montreal
name: Montreal
state: QC
country: CA
population: 1704694
area: 431.50
safety: 7.80
density: 3950.62
livability: 4.73
Total number of keys: 8
```