

Control

Recap

- `if` / `elif` / `else`
- comparison operators (`>` , `>=` , ...)
- logical operators (`and` , `or` , `not`)
- boolean values & expressions (`2 > 1` , `True` , `False`)
- control flow using `return`

Recap

```
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")
```

Loop

`while`

`for`

while: conditionally repeated

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


while using -= for assignment

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

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

range(start, end)

```
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)
```


3. 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 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")
```

```
def main():  
    # Ask the user to enter a positive number  
    n = get_positive_number()  
  
    # Print "meow" n times  
    meow(n)  
  
main()
```

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()
```



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
```




4. 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`)

- add
- delete
- join
- search
- loop
- sort

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"]
```

Join two lists - `+`, `extend`

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

cities3 = cities + cities2
print(cities3)

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

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])
```

Sort list - `sort`

```
cities = ["Montreal", "Toronto", "Vancouver", "Detroit"]  
  
cities.sort()  
print(cities)  
  
cities.sort(reverse=True)  
print(cities)
```

dictionary: 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)
```

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

```
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)
```

`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)
```

Sort dict - sorted

```
cities = {  
    "name": "Montreal",  
    "state": "QC",  
    "country": "CA"  
}  
  
sorted(cities.items())
```

	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>



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
```



6. 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))
```


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

```
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"}  
]
```

`cities`

```
[  
    {...}, # index 0  
    {...}, # index 1  
    {...}, # index 2  
    {...}  # index 3  
]
```

`cities[0]`

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

`cities[0]["state"]`

```
"QC"
```

list of dict

```
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"])
```

list of list

```
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"])
```

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: new york
New york is not available in data
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
```


Quiz prep

1. Review the slides
2. Mock exam (Under "Coursework" on myCourses)
3. Post (and answer) questions on Ed Discussion
4. Q&A (15 min before the quiz)