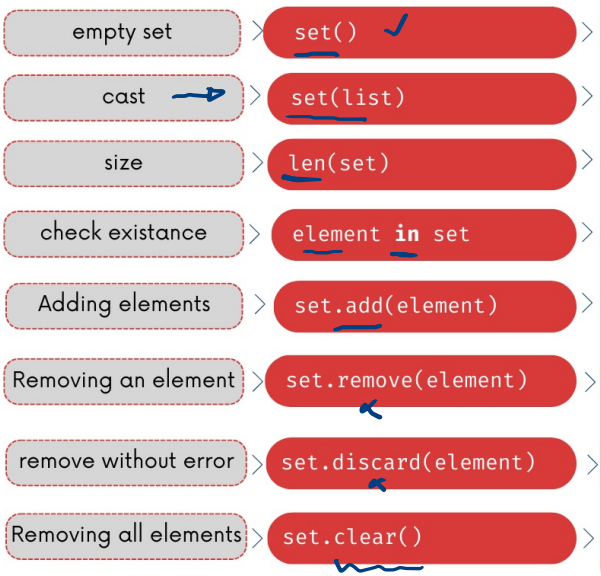
python / session 2

Set

- A Python set is an unordered list of immutable elements. It means:
  - Elements in a set are unordered.
  - Elements in a set are unique. A set doesn't allow duplicate elements.
  - Elements in a set cannot be changed. For example, they can be numbers, strings, and tuples, but cannot be lists or dictionaries.

```
{'r', 'l', 't', 'e'}  #curly brace {}
```

Set Manipulation




```
letters = set(['P','C'])
print(letters)
#out: {'P', 'C'}

ratings = {1, 2, 3, 4, 5}
size = len(ratings)
print(size)
#out: 5

ratings = {1, 2, 3, 4, 5}
rating = 1
if rating in ratings:
   print('there is')
#out: there is

s = {'P', 'S', 'P'}
s.remove('S')
```

String → Iterable

Strings & Loops

- these two code snippets do the same thing

```
>>> s = "abcdefgh"

>>> for index in range(len(s)):
>>>     if s[index] == 'i' or s[index] == 'u':
>>>         print("There is an i or u")



>>> for char in s:
>>>     if char == 'i' or char == 'u':
>>>         print("There is an i or u")
```

The f-strings provide a way to embed variables and expressions inside a string literal using a clearer syntax than the format() method.

F-strings

- F'text {first_variable} text {second_variable}' --> F or f

```
s = F'Hello, {first_name} {last_name}!'
```

- Multiline F-String

```
message = ( f'Hello {name}. '
f"You're learning Python at {website}.")

message = f'Hello {name}. ' \
f"You're learning Python at {website}."

message = f"""Hello {name}. You're
learning Python at {website}."""
```

F-strings

- Format numbers using f-strings

```
number = 200
 s = f'{number: 06}'
print(s)
#space06
# 00200, pad zeros
# at the beginning of the number
```

```
number = 9.98567
 s = f'{number: .2f}'
print(s) # 9.99
```

```
number = 400000000000
 s = f'{number: ,}'
 # also can use _
print(s) # 400,000,000,000
```

```
number = 0.1259
s = f'{number: .2%}'
print(s)
# 12.59%
s = f'{number: .1%}'
print(s) # 12.5%
```

Raw strings

In Python, when you prefix a string with the letter r or R such as r'...' and R'...', that string becomes a raw string.

```
s ='lang\tver\nPython\t3'
print(s)

lang ver
Python 3
```

```
s = r'lang\tver\nPython\t3'
print(s)

#out: lang\tver\nPython\t3
```

Backslash

- #backslash (\) escape other special characters
- \n: new line
- \t: Tab means(8 spaces)
- \\: Back slash
- \': Single quote (')
- \": Double quote (")

```
print('I suppose you are student.\nAre you ?')

#I suppose you are student.
#Are you ?
```

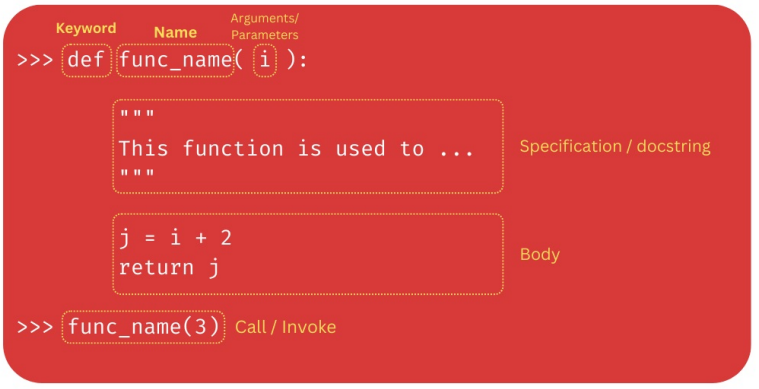
```
print('I suppose you are student.\\nAre you ?')

#I suppose you are student.\nAre you ?
```

```
print('Days\tTopics\tExercises')

#Days Topics Exercises
```

```
print('Days\\tTopics\\tExercises')

#Days\tTopics\tExercises
```

Python Function

A function is a named code block that performs a job or returns a value.

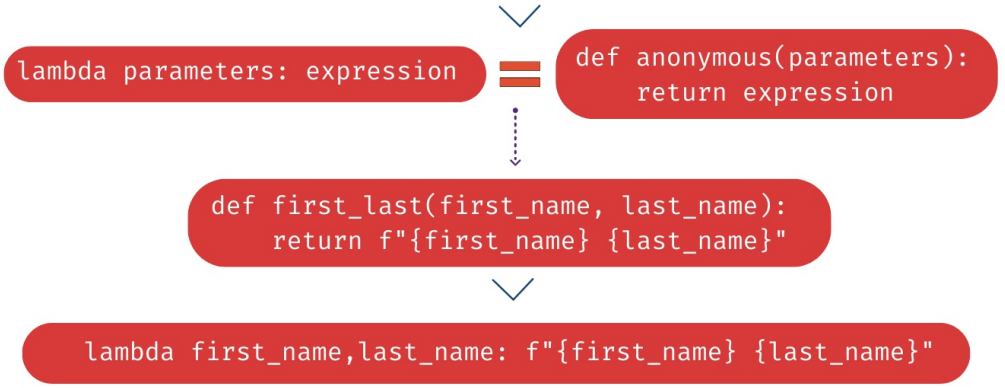


Lambda Expressions

Python lambda expressions allow you to define anonymous functions.

Anonymous functions are functions without names. The anonymous functions are useful when you need to use them once.

A lambda expression typically contains one or more arguments, but it can have only one expression.

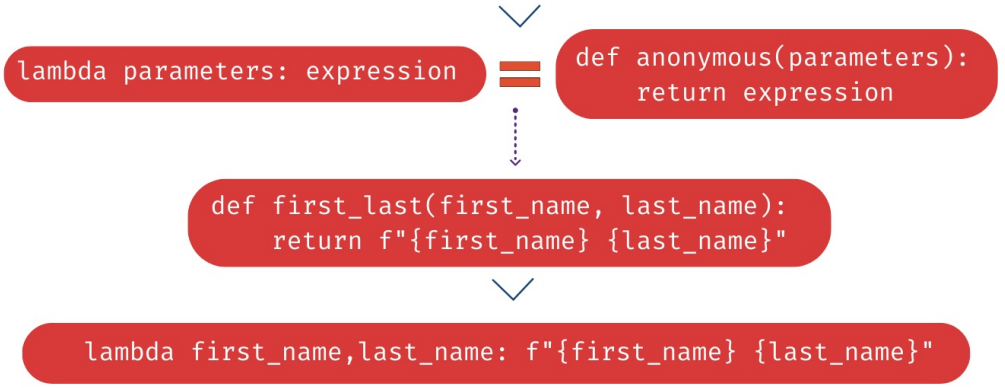


Lambda Expressions

Python lambda expressions allow you to define anonymous functions.

Anonymous functions are functions without names. The anonymous functions are useful when you need to use them once.

A lambda expression typically contains one or more arguments, but it can have only one expression.



Lambda Examples

```
x = lambda a : a + 10
print(x(5))
```

```
x = lambda a, b : a * b
print(x(5, 6))
```

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
print(mydoubler(11))
```

src: w3schools.com

ITERATION vs RECURSION

ITERATION

```
# factorial iteration solution
>>> def fact_iter(n):
        prod = 1
        for i in range(1, n+1):
            prod *= i
        return prod

>>> fact_iter(5)
```

RECURSION

```
# factorial recursive solution
>>> def fact_recur(n):
        if n == 1:
            return 1
        else:
            return n * fact_recur(n-1)

>>> fact_recur(n)
```

- recursion may be simpler, more intuitive
- recursion may be efficient from programmer POV
- recursion may not be efficient from computer POV

for element in list:

while, wile

type of errors

- **Syntax Error** > When you write an invalid Python code, you'll get a syntax error.
- **Exceptions** > In Python, errors that occur during the execution are called exceptions.

```
current = 1
if current < 10
current += 1
```

Syntax Error

```
print(0/0)
```

Exception Error

string

func (a, b)

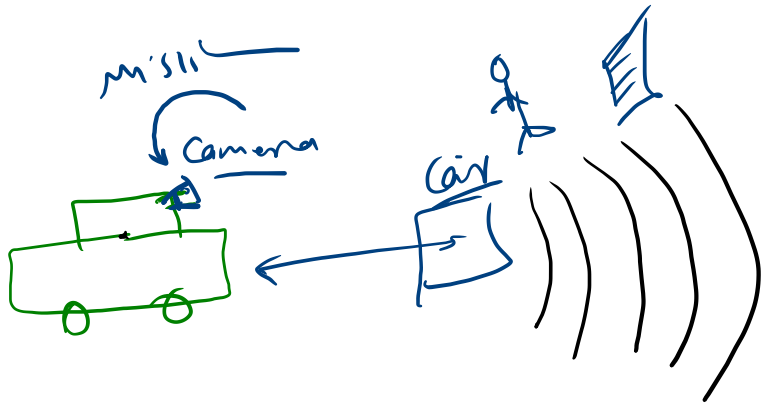
print (a + b) → 5

Exception

Few : Forward Collision Warning!

Monocular

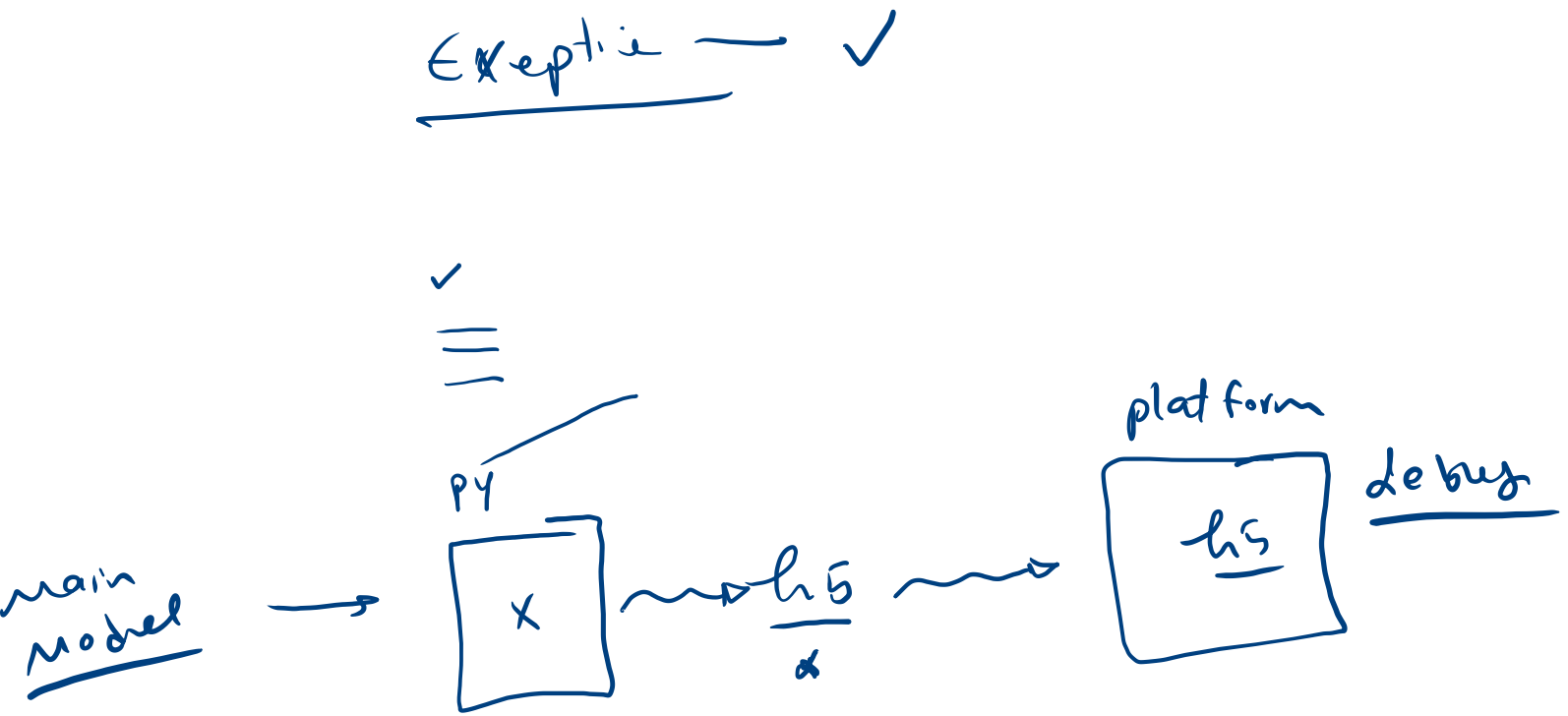
Azil



model → model

empty frames

Ex —


Exeptio
plat form
debug
py
main
model
x
h5
h5

try...except

The try block lets you test a block of code for errors.

The except block lets you handle the error.

```
try:
# code that may cause error

except:
# handle errors
```

```
def division(a, b)
    try:
       print(a/b)

    except:
       print("An exception occurred")
```

src: w3schools.com

→ $a \times b$

→ $\underbrace{a + a + a + a \cdots + a}_{b}$