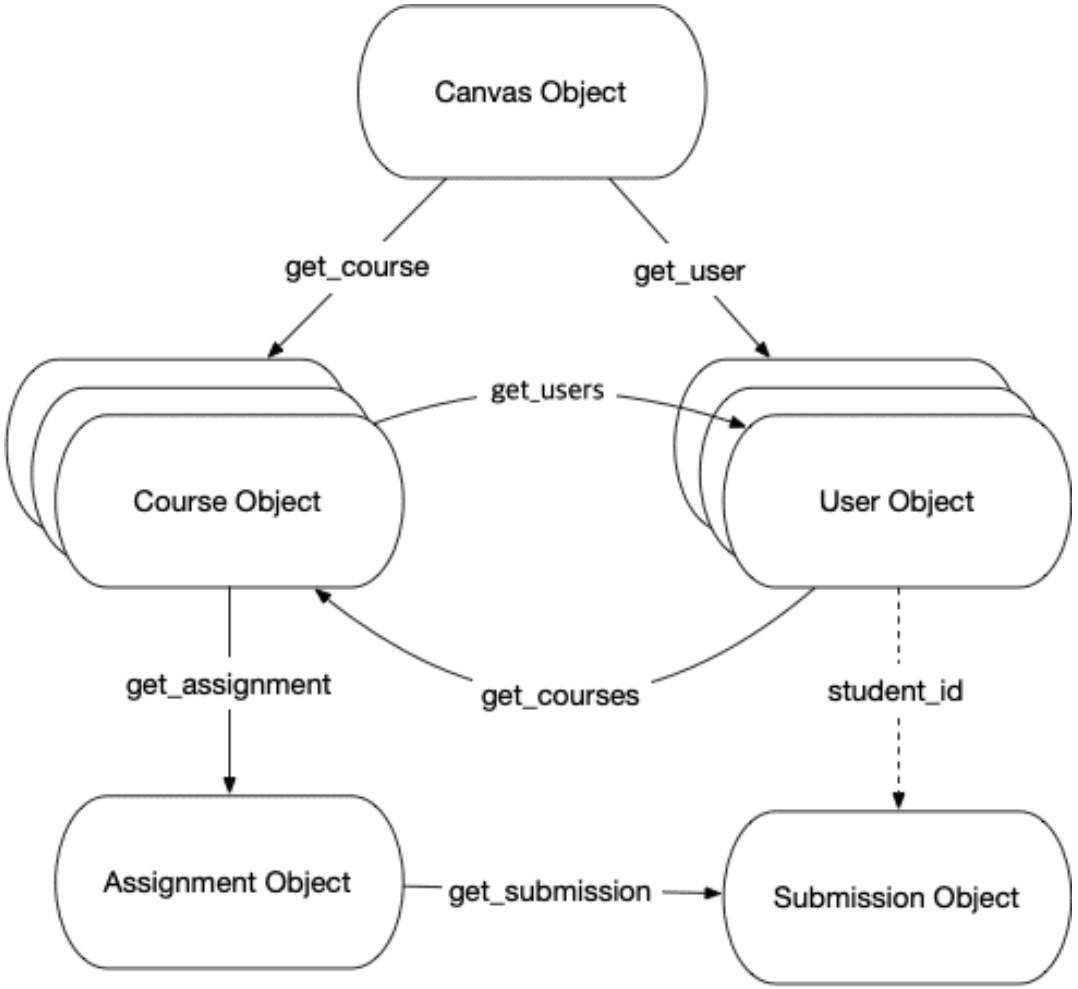
Getting Started with CanvasAPI

https://canvasapi.readthedocs.io/en/stable/getting-started.html

Objects


Canvas Object
get_course
get_user
get_users
Course Object
User Object
get_assignment
get_courses
student_id
Assignment Object
get_submission
Submission Object

```
# Canvas object
canvas = Canvas(API_URL, API_KEY)

# Course object
course = canvas.get_course(123456)
course.name
course.update(course={'name': 'New Course Name'})

# User object
user = canvas.get_user(123)
login_id_user = canvas.get_user('some_user', 'sis_login_id')

user.name
courses = user.get_courses()
```

Warning: PaginatedList

PaginatedList lazily loads its elements. Unfortunately, there's no way to determine the exact number of records Canvas will return without traversing the list fully.

```
courses = user.get_courses()
print(courses) => <PaginatedList of type Course> # This is an issue
for course in courses:
    print(course)
```

We need to force iteration so the library actually makes all API calls

```
courses = list(user.get_courses())
print(len(courses))
```

DateTimes

We can use Python's datetime in canvasapi

```
from datetime import datetime

start_date = datetime(2018, 1, 1, 0, 1)  # '2018-01-01T00:01Z'
end_date = datetime(2018, 12, 31, 11, 59)  # '2018-12-31T11:59Z'
course.update(
    course={
        'start_at': start_date,
        'end_at': end_date,
    }
)
```

If Canvas returns an ISO 8601 formatted datetime string, CanvasAPI will automatically create a datetime object from that string.

```
course = canvas.get_course(1)
course.start_at # '2014-02-11T16:38:00Z'
course.start_at_date # datetime.datetime(2014, 2, 11, 16, 38, tzinfo=<UTC>)
```

User Object

Type

```
users = course.get_users()

# Get all students
for user in users:
    print(user)

# Get only students
users = course.get_users(enrollment_type=['student'])

# Get teachers, tas, and designers
type_list = ['teacher', 'ta', 'designer']
users = course.get_users(enrollment_type=type_list)
```

State

```
users = course.get_users(
    enrollment_type=['teacher', 'student']
    enrollment_state=['active', 'invited']
)
```

Assignments

Get all or ungraded assignments.

```
assignments = course.get_assignments()
assignments = course.get_assignments(bucket='ungraded')
```

Create a new assignment.

```
new_assignment = course.create_assignment({
    'name': 'Assignment 1'
})
new_assignment = course.create_assignment({
    'name': 'Assignment 2',
    'submission_types': ['online_upload', 'online_text_entry', 'online_url']
})
new_assignment = course.create_assignment({
    'name': 'Assignment 3',
    'submission_types': ['online_upload'],
    'allowed_extensions': ['docx', 'doc', 'pdf'],
    'notify_of_update': True,
    'points_possible': 100,
    'due_at': datetime(2018, 12, 31, 11, 59),
    'description': 'Please upload your homework as a Word document or PDF.',
    'published': True
})
```

Submission

```
submission = assignment.get_submission(student_id)
submission.edit(submission={'posted_grade':score})
```

Example: Add n Points to All Users' Assignment

```
added_points = 2

submissions = assignment.get_submissions()
for submission in submissions:
    if submission.score is not None:
        score = submission.score + added_points
    else:
        score = 0 + added_points
    submission.edit(submission={'posted_grade': score})
```