

공학적 문제해결 기법

11주차 2차시

객체와 클래스에 대한 그래픽적 접근





11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

I. 개요

그림속의 객체와 클래스 및 유용한 내장함수들

II. 학습 개요

1) 학습 목표

turtle 모듈을 이용하여 객체와 클래스에 대해 그래픽적으로 접근해보고, 유용한 내장함수들을 알아본다.

2) 학습 목차(세부 목차)

- 그림속의 객체와 클래스
- 유용한 내장함수들



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 1 - 그림속의 객체와 클래스

그래픽적 접근

- ✓ 앞서 배운 turtle 모듈로 돌아가보자
- ✓ turtle.Pen()을 이용하면 turtle 모듈이 제공하는 Pen 클래스의 객체 생성 가능
- ✓ 두 개의 기린을 생성한 것처럼 avery와 kate라는 이름의 거북이 객체 두 개를 생



성할 수 있다.

```
>>> import turtle
```

```
>>> avery = turtle.Pen()
```

```
>>> kate = turtle.Pen()
```

- ✓ 각각의 거북이 객체는 (avery와 kate) Pen 클래스의 멤버
- ✓ 거북이 객체를 생성했다면 각 객체에서 함수를 호출할 수 있고,
- ✓ 독립적으로 그림을 그릴 수 있다.



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 1 - 그림속의 객체와 클래스

그래픽적 접근

☒ 다음과 같이 해보자.

➡

```
>>> avery.forward(50)
>>> avery.right(90)
>>> avery.forward(20)
```

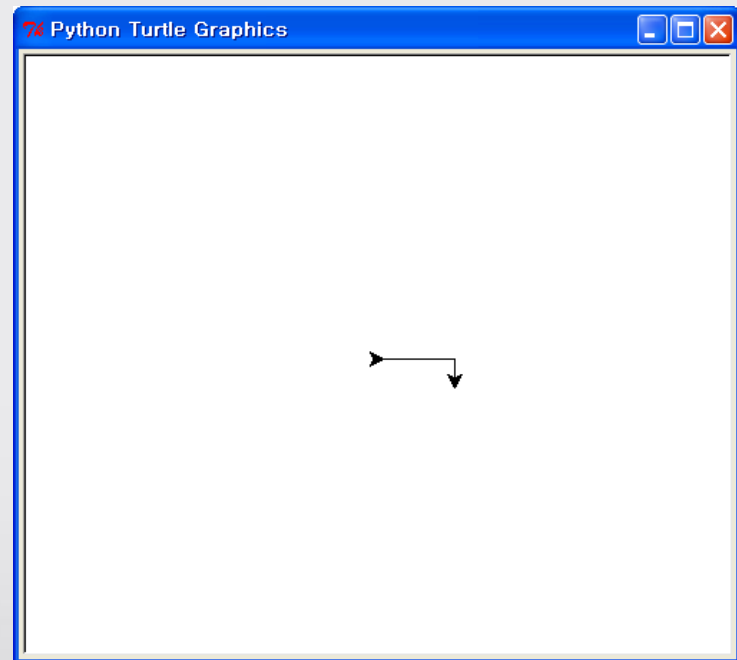
☒ avery에게 50픽셀 앞으로 움직이고.

☒ 오른쪽으로 90도 회전

☒ 다음에 20픽셀 앞으로 이동

☒ avery는 화면의 아래를 바라보면서 멈춤

☒ 거북이는 맨 처음에 시작될 때는 항상 오른쪽 방향을 향하고 있다.





11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 1 - 그림속의 객체와 클래스

그래픽적 접근

☒ 이제 kate가 움직일 차례

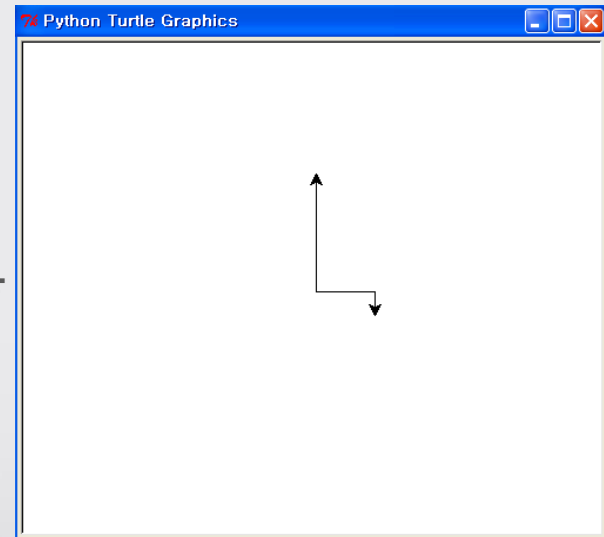


```
>>> kate.left(90)
```

```
>>> kate.forward(100)
```

☒ kate에게 왼쪽으로 90도 돌고 나서 100픽셀 앞으로 이동

☒ kate는 화면의 위쪽을 향함






11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 1 - 그림속의 객체와 클래스

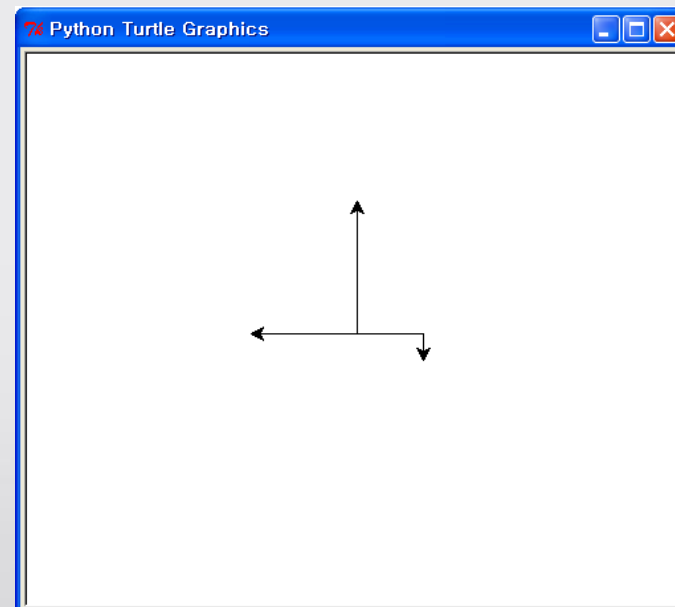
그래픽적 접근

☒ 이제 또 다른 거북이인 jacob을 추가하고 이것도 움직여 보자.



```
>>> jacob = turtle.Pen()  
>>> jacob.left(180)  
>>> jacob.forward(80)
```

☒ 이것을 그리면 다음과 같다.





11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 1 - 그림속의 객체와 클래스

그래픽적 접근

- ✓ 거북이를 생성하기 위해서 `turtle.Pen()`을 호출할 때마다 새롭고 독립적인 객체를 추가하게 된다
- ✓ 각 객체는 여전히 `Pen` 클래스의 인스턴스
- ✓ 각 객체에 있는 동일한 함수를 사용할 수 있다
- ✓ 실습으로 확인했듯이 각 객체들은 독립적으로 움직일 수 있다.
- ✓ 이미 생성했던 객체와 동일한 변수 이름을 가진 새로운 객체를 생성한다면?
- ✓ 이전 객체가 반드시 사라지지 않을 수 있다.
- ✓ 또 하나의 `kate` 거북이를 생성하고 움직여보자. 결과는?



```
>>> import turtle  
>>> avery = turtle.Pen()  
>>> kate = turtle.Pen()
```



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들

내장함수

- ☒ 모듈은 사용하기 전에 임포트해야 하지만,
내장 함수는 파이썬 셸을 시작하자마자 사용 가능
- ☒ 내장된 도구(정말로 거대한 코드)는 프로그램을 훨씬 쉽게 작성하도록 해 줌



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들

ABS 함수

- ✓ 숫자의 절대값(부호가 없는 숫자값)을 반환
- ✓ 10의 절대값은 10이며, -10의 절대값도 10

➡

```
>>> print(abs(10))  
>>> print(abs(-10))
```

- ✓ 게임에 있는 캐릭터가 방향에 상관없이 얼마나 움직였는지를 계산할 때 유용

➡

```
>>> steps = -3  
>>> if abs(steps) > 0:  
    print('Character is moving')
```



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들

bool 함수

- ✓ 단 하나의 매개변수를 받으며 그 값으로 True 또는 False를 반환
- ✓ 숫자에 bool을 사용하면 0은 False를 반환하고, 그 외의 값은 True를 반환



```
>>> print(bool(0))
```

```
False
```

```
>>> print(bool(1123.23))
```

```
True
```

- ✓ 문자열에 값이 없을 경우 (다시 말해서 None, 즉 빈 문자열)에 False를 반환, 그렇지 않으면 True를 반환
- ✓ 아무런 값을 가지고 있지 않은 리스트와 맵에 대해 False를 반환, 값을 가지고 있다면 True를 반환
- ✓ 값이 설정되어있는지 아닌지를 결정해야 할 경우에 bool을 사용



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들

bool 함수

 예)




```
>>> year = input('Year of birth: ')
```

```
Year of birth:
```

```
>>> if not bool(year.rstrip()) :
```

```
    print('You need to enter a value for your year of birth')
```

 rstrip 함수 : 모든 공백과 문자열의 끝에 있는 엔터 문자를 제거하는 함수

 사용자가 아무것도 입력하지 않으면,

 if 문은 not을 사용하고 있으므로 거짓의 반대인 참이 됨

 print문 안의 내용이 출력될 것임



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들

`eval`(`evaluate`의 약자) 함수

☒ 매개변수로 문자열을 받음

☒ 표현식인 것처럼 실행

➡

```
>>> eval('10*5')  
50
```

☒ 사용자의 입력을 표현식으로 바꿀 때 종종 사용

☒ 사용자 입력은 문자열로 읽혀짐

☒ 계산을 하기 전에 그것을 숫자와 연산자로 변환해야 하는데, `eval` 함수는 이러한 변환을 해줌

➡

```
>>> your_calculation = input('Enter a calculation : ')  
Enter a calculation : 12 * 52  
>>> eval(your_calculation)  
624
```



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들

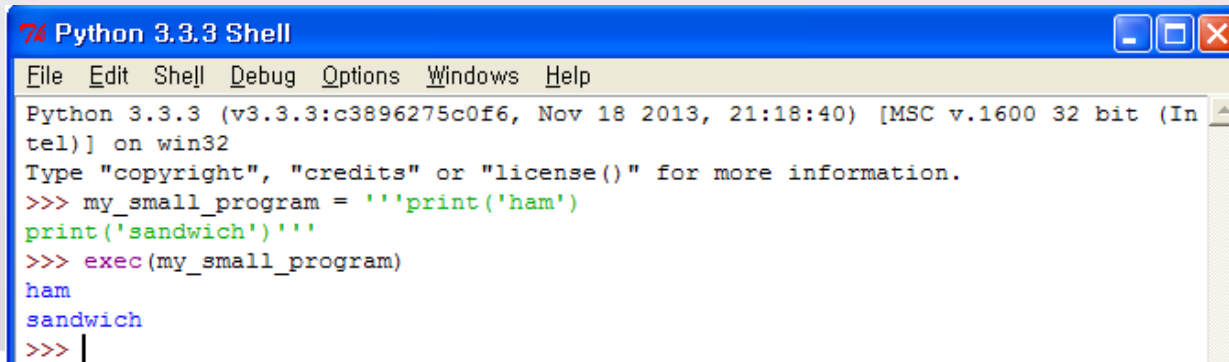
exec 함수

- ✓ eval 함수와 유사하나, 좀 더 복잡한 프로그램들을 실행할 때 사용될 수 있다.

➡

```
>>> my_small_program = '''print('ham')
print('sandwich')'''
>>> exec(my_small_program)
```

- ✓ 파일로 읽어 들인 작은 프로그램을 실행하려고 할 때 exec를 사용
- ✓ 프로그램속의 프로그램



```
Python 3.3.3 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2013, 21:18:40) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> my_small_program = '''print('ham')
print('sandwich')'''
>>> exec(my_small_program)
ham
sandwich
>>> |
```



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들

float, int, len 함수

- ☑ float 함수는 문자열이나 숫자를 실수 (real number)라고 하는 소수점이 있는 부동 소수점 (floating-point) 숫자로 변환

➡ >>> float('12')
12.0

- ☑ int 함수는 문자열이나 숫자를 범자연수 (즉, 정수)로 변환, 기본적으로 소수점 이하의 모든 것들을 버림




11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들


float, int, len 함수

 len 함수는 객체(문자열)의 길이(글자 개수)를 반환

 리스트나 튜플에서 사용하면 거기에 있는 항목들의 개수를 반환

 맵에서 사용하면 맵에 있는 항목들의 개수를 반환

 루프(loop) 작업을 할 때 특히 유용, 항목들의 인덱스 위치를 표시



```
>>> creature_list = [ 'unicorn', 'cyclops', 'fairy', 'elf', 'dragon',  
    'troll' ]  
  
>>> print(len(creature_list))
```




11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들

min, max 함수

- ✓ max 함수는 리스트, 튜플 또는 문자열에 있는 가장 큰 항목을 반환

➡

```
>>> numbers = [ 5, 4, 10, 30, 22 ]  
>>> print(max(numbers))  
30
```

- ✓ min 함수는 max 함수처럼 동작하지만, 리스트나 튜플 또는 문자열에 있는 가장 작은 항목을 반환

➡

```
>>> numbers = [ 5, 4, 10, 30, 22 ]  
>>> print(min(numbers))  
4
```

- ✓ 비교하고자 하는 항목들을 매개변수처럼 괄호 안에 넣어서 사용 가능

➡

```
>>> print(min(300, 10, 450, 50, 90))  
10
```



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들

max 함수 예

네 명의 플레이어로 구성된 팀과 여러분이 숫자 맞추기 게임을 하고 있다고 가정하자. 플레이어들이 생각한 숫자는 여러분이 생각한 숫자보다 작아야 하는 게임이다. 만약 플레이어들이 생각하는 숫자가 여러분의 숫자보다 크면 지게 되는 것이고, 그렇지 않으면 이기는 것이다.

 이 모든 숫자들이 높은지 낮은지를 빠르게 찾기 위해서 max 함수를 사용

 간단하게 만들어보면,

```
➡ >>> guess_this_number = 61
>>> player_guesses = [ 12, 15, 70, 45 ]
>>> if max(player_guesses) > guess_this_number:
    print('Boom! You all lose')
else:
    print('You win')
```



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들

range 함수

- ✓ 특정 횟수만큼 코드를 반복하는 for 루프에 주로 사용
- ✓ range 함수에 주어지는 처음 두 개의 매개변수는 시작과 끝이라고 부름

➡

```
>>> print(list(range(0, 5)))
```



```
[0, 1, 2, 3, 4]
```

- ✓ range에 증감값(step)이라는 세 번째 매개변수를 추가 가능

➡

```
>>> count_by_twos = list(range(0, 30, 2))
```



```
>>> print(count_by_twos)
```



```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

```
>>> count_down_by_twos = list(range(40, 10, -2))
```



```
>>> print(count_down_by_twos)
```



```
[40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12]
```



11주차 2차시 : 객체와 클래스에 대한 그래픽적 접근

III. 학습 2 - 유용한 내장함수들

sum 함수

 리스트에 있는 항목들을 더해서 그 합계를 반환

➡

```
>>> my_list_of_numbers = list(range(0, 500, 50))  
>>> print(my_list_of_numbers)  
[0, 50, 100, 150, 200, 250, 300, 350, 400, 450]  
>>> print(sum(my_list_of_numbers))  
2250
```



『 이 콘텐츠는 2014학년도 학부교육 선도대학 육성사업(ACE)에 의하여 개발한 것임 』

