CPE/CSC 101

Today

- while
- decomposition / reuse examples
- strings

- Quiz

image.ppm

P3
width height
max color value
pixels

print('P3')
print(width, height)
print(255)

~ cast_all_rays

while

```
def sum(L: List[int]) -> int:
    result = 0
    for n in L:
        result += n
    return result
```

```
for i in range(len(L)):
    result += L[i]
```

```
def sum(L: List[int]) -> int:
    result = 0
    i = 0
    while i < len(L):
        result += L[i]
        i += 1
    return result
```

```
while cond:
    body
```

```
if (B ** 2 - 4 * A * C) < 0:
    return None
else:
    t1 = (-B + math.sqrt(B ** 2 - 4 * A * C)) / (2 * A)
    t2 = (-B - math.sqrt(B ** 2 - 4 * A * C)) / (2 * A)

    point_t1 = vector_math.translate_point(the_ray.pt, vector_math.scale_vector(the_ray.dir, t1))
    point_t2 = vector_math.translate_point(the_ray.pt, vector_math.scale_vector(the_ray.dir, t2))


# if the ray goes through the sphere intersecting at two points

    if t1 >= 0 and t2 >= 0:
        if t1 < t2:
            return point_t1
        else:
            return point_t2
    # if the ray is inside the sphere
    elif t1 >= 0 or t2 >= 0:
        if t1 >= 0 and t2 < 0:
            return point_t1
        else:
            return point_t2
    # if the ray is not intersecting the sphere
    else:
        return None
```

```python
def t_value(theRay: data.Ray, theSphere: data.Sphere) -> Optional[float]:
    A = vector_math.dot_vector(theRay.dir, theRay.dir)
    B = (2 * vector_math.dot_vector(vector_math.vector_from_to(theSphere.center, theRay.pt), theRay.dir))
    C = ((vector_math.dot_vector(vector_math.vector_from_to(theSphere.center, theRay.pt), vector_math.vector_from_to(theSphere.center, theRa

    if B**2 - 4 * A * C < 0:  # if it's negative, we're done
        return None

    t1 = (-B + math.sqrt(B**2 - 4 * A * C)) / (2 * A)
    t2 = (-B - math.sqrt(B**2 - 4 * A * C)) / (2 * A)

    if t1 >= 0 and t2 >= 0:
        return min(t1, t2)
    elif t1 >= 0:
        return t1
    elif t2 >= 0:
        return t2
    else:
        return None
```

```python
def point_t(theRay: data.Ray, t: float) -> data.Point:
    return vector_math.translate_point(theRay.pt, vector_math.scale_vector(theRay.dir, t))
```

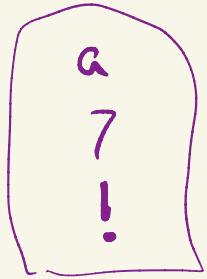
```python
def sphere_intersection_point(the_ray: data.Ray,
                              sphere: data.Sphere) -> Optional[data.Point]:
    t = t_value(the_ray, sphere)  # t as float or none
    if t is not None:
        return point_t(the_ray, t)
    else:
        return None
```

PyCharm Example

- from Lab 6

Strings

sequence of characters



'a'
'7'
'!'

"a"
"7"
"!"

'hello'

'hello' + 'there'
⇒ 'hellothere'

`'hello' + str(7)`

=> `'hello7'`

```
s = 'hello'
s2 = 'hi'

s == s2
  => False

s == 'hello'
  => True
```

```
if __name__ == '__main__':
    main()
```

'hello' < 'bye'

⇒ False

(104) hello

bye (98)

'by' < 'bye'

⇒ True

b y

b y e

ord('a') → 97

chr(98) → 'b'

based on some encoding of characters

ASCII

'a' → 97

allowed 0-127

extended 0-255

UNICODE

String is a sequence
- similar to a list
- immutable

```
s = 'hello'
```

s[1]

'hello'

```
for c in s:
    print(c)
```

```
h
e
l
l
o
```

my_join(['a', 'b', 'c'], ',')

=> 'a,b,c'

```
def my_join(strings: List[str], delim: str) -> str:
    result = ''
    for word in strings:
        result = result + word + delim

    return result
```

```
result = ''
result = 'a,'
result = 'a,b,'
result = 'a,b,c,'
```

```
def my_join(strings: List[str], delim: str) -> str:
    if strings:                          # if list is not empty
        result = strings[0]
        for i in range(1, len(strings)):
            result = (result + delim + strings[i])
        return result
    else:
        return ''
```

result = 'a'

result = 'a, b'

result = 'a, b, c'