Python Foundations

Guttag Chapter 2

Prof. Graber

Goals

- Review Python literals
- Review Python variables and operators
- Review Python loops and conditionals
- Review Python squaring algorithms
- Review Python strings and slicing

Python Literals

String

```
print("Hello")
print(type("Hello"))
```

```
Hello
<class 'str'>
```

. . .

TODO: print your name below

. . .

Integer

```
print(101)
print(type(101))
```

```
101
<class 'int'>
```

. . .

TODO: print an integer literal that is negative

. . .

Float

```
print(101.1)
print(type(101.1))
```

```
101.1
<class 'float'>
```

. . .

TODO: print a float literal that is negative

. . .

Boolean

```
print(True)
print(type(True))
```

```
True
<class 'bool'>
```

. . .

TODO: print a boolean literal that is not `True`

. . .

List

```
print([False, -12, -34.1, "literals!"])
print(type([False, -12, -34.1, "literals!"]))
```

```
[False, -12, -34.1, 'literals!']
<class 'list'>
```

. . .

TODO: print a list literal that has 3 integers

. . .

Tuple

```
print((False, -12, -34.1, "literals!"))
print(type((False, -12, -34.1, "literals!")))
```

```
(False, -12, -34.1, 'literals!')
<class 'tuple'>
```

. . .

TODO: print a tuple literal that has 3 floats

. . .

Python Variables and Operators

Assignment

```
a = [False, -12, -34.1, "assignment!"]
print(a)
print(type(a))
```

```
[False, -12, -34.1, 'assignment!']
<class 'list'>
```

- `[False, -12, -34.1, "assignment!"]` is the object in memory
- `a` is the variable name given to the object
- `=` operator used
- variable name always goes on the left hand side of `=`

Arithmetic

```
item1 = 100
item2 = 11
addition_ex = item1 + item2
mult_ex = item1 * item2
div_ex = item1 / item2
floor_div_ex = item1 // item2
power_ex = item2 ** 2
mod_ex = item2 % 7

print(f"addition_ex is {addition_ex}")
print(f"mult_ex is {mult_ex}")
print(f"div_ex is {div_ex}")
print(f"floor_div_ex is {floor_div_ex}")
print(f"power_ex is {power_ex}")
print(f"mod_ex is {mod_ex}")
```

```
addition_ex is 111
mult_ex is 1100
div_ex is 9.090909090909092
floor_div_ex is 9
power_ex is 121
mod_ex is 4
```

Comparison

```python
item1 = 100
item2 = 11
gt = item1 > item2
lt = item1 < item2
gte = item1 >= item2
lte = item1 <= item2
equal = item1 == item2
not_equal = item1 != item2

print(f"item1 > item2 is {gt}")
print(f"item1 < item2 is {lt}")
print(f"item1 >= item2 is {gte}")
print(f"item1 <= item2 is {lte}")
print(f"item1 == item2 is {equal}")
print(f"item1 != item2 is {not_equal}")
```

```
item1 > item2 is True
item1 < item2 is False
item1 >= item2 is True
item1 <= item2 is False
item1 == item2 is False
item1 != item2 is True
```

Python Loops and Conditionals

For Loop

```python
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
```

```
6
7
8
9
```

For Loop

```
for i in range(10):
    print(i)
```

TODO: Write a for loop to print out the square root of every even integer between 2 and 64, including 64

. . .

While Loop

```
i = 0
while i < 10:
    print(i)
    i += 1
```

```
0
1
2
3
4
5
6
7
8
9
```

While Loop

```
i = 0
while i < 10:
    print(i)
    i += 1
```

TODO: What happens if you forget to increment the counter?

. . .

Conditional

```
1 i = 0
2 while i < 10:
3     if i % 2 == 1:
4         print(f"{i} is odd!")
5     elif i % 3 == 0:
6         print(f"{i} is divisible by 3!")
7     i += 1
```

```
0 is divisible by 3!
1 is odd!
3 is odd!
5 is odd!
6 is divisible by 3!
7 is odd!
9 is odd!
```

Conditional

TODO: Write one loop that prints “fizz” for every even number, “buzz” for every multiple of 5, and “fizzbuzz” for even multiples of 5 for integers from 0 to 20 including 20.

. . .

Python Squaring Algorithms

Square by Addition in a For Loop

```
# choose a value to square
value = 6
# initialize the answer
answer = 0
# repeatedly increase the answer until getting to the value
for _ in range(value):
    answer += value
# print the computed integer squared using a for loop
print(f"{value} squared is {answer}")
```

```
6 squared is 36
```

Square by Addition in a While Loop

```
# choose a value to square
value = 7
# initialize the number of iterations and the answer
i = 0
answer = 0
# repeatedly increase the answer until getting to the value
while i < value:
    answer += value
    i += 1
# print the computed integer squared using a while loop
print(f"{value} squared is {answer}")
```

```
7 squared is 49
```

Abstracting to a Function

```python
# choose a value to square
def square_by_addition_while(value: int) -> int:
    """Square a number by addition in a while loop."""
    # initialize the number of iterations and the answer
    i = 0
    answer = 0
    # repeatedly increase the answer until getting to the value
    while i < value:
        answer += value
        i += 1
    # return the computed integer squared using a while loop
    return answer
```

Abstracting to a Function

```python
# choose a value to square
def square_by_addition_while(value: int) -> int:
    """Square a number by addition in a while loop."""
    # initialize the number of iterations and the answer
    i = 0
    answer = 0
    # repeatedly increase the answer until getting to the value
    while i < value:
        answer += value
        i += 1
    # return the computed integer squared using a while loop
    return answer
```

Notice:

- keyword `def`
- indentation
- type annotations for input and output `: int` and `-> int`
- docstring `"""Square..."""`
- return statement `return`

TODO: Why didn't anything print?

Abstracting to a Function

```
# choose a value to square
def square_by_addition_while(value: int) -> int:
    """Square a number by addition in a while loop."""
    # initialize the number of iterations and the answer
    i = 0
    answer = 0
    # repeatedly increase the answer until getting to the value
    while i < value:
        answer += value
        i += 1
    # return the computed integer squared using a while loop
    return answer

my_value = 12
my_answer = square_by_addition_while(my_value)
print(f"{my_value} squared is {my_answer}")
```

```
12 squared is 144
```

. . .

TODO: What happens if you try to print `print(f"{value} squared is {answer}")`? Copy the code and then move to the next slide.

Abstracting to a Function

TODO: What happens if you try to print `print(f"{value} squared is {answer}")`? Paste the code and try it.

Python Strings and Slicing

Formatted Strings

```python
value = 12345
formatted_string = f"the value is {value}"
print(formatted_string)
```

```
the value is 12345
```

Slicing

```python
value = 12345
formatted_string = f"the value is {value}"
print(formatted_string)
print(formatted_string[4:14:1])
```

```
the value is 12345
value is 1
```

. . .

The indices are the starting point, the end point (non-inclusive), the hop

Find Further Review in Chapter 2