Python Functions

CS106AP Lecture 6
Everyday Python

Object-Oriented Programming

Data structures

Midterm

Graphics

Programming Basics

The Console

Strings and the Console

Python Functions

Images

Life after CS106AP!
How do we translate what we know from Karel into regular Python code?

How can we make our code more flexible by producing different outputs depending on the input?
Today’s topics

1. Introduction and Review
2. Range For Loops
3. Python Functions
4. Variable Scope
5. What’s next?
Who am I?
Sonja Johnson-Yu
Sonja
Johnson-Yu
Sonja Johnson-Yu
Sonja Johnson-Yu
Review
Variables
What is a variable?

A variable is a container for storing a data value.

\[ \text{num\_flowers} = 5 \]
What is a variable?

A variable is a container for storing a data value.

```
num_flowers = 5
```

variable's value
Terminology summary

- Variables have a **name** and are associated with a **value**

- Variable **assignment** is the process of associating a value with the name (use the equals sign =)

- **Retrieval** is the process of getting the value associated with the name (use the variable’s name)
  - This is how you use variables!
Expressions
Recall: expressions

- The computer **evaluates** expressions to a single value
- We use **operators** to combine literals and/or variables into **expressions**
Arithmetic operators

* Multiplication
/ Division
// Integer division
% Modulus (remainder)
+ Addition
- Subtraction

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>1</td>
</tr>
<tr>
<td>*, /, //, %</td>
<td>2</td>
</tr>
<tr>
<td>+, −</td>
<td>3</td>
</tr>
</tbody>
</table>
Arithmetic operators

* Multiplication
/ Division
// Integer division
% Modulus (remainder)
+ Addition
- Subtraction

Integer division takes the largest integer that is equal to or smaller than the quotient

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>1</td>
</tr>
<tr>
<td>*, /, //, %</td>
<td>2</td>
</tr>
<tr>
<td>+, -</td>
<td>3</td>
</tr>
</tbody>
</table>
Integer Division Practice!

- $5 + 1 \div 2$
- $9 \div 3$
- $8 \div 3$
- $-8 \div 3$

Integer division takes the largest integer that is equal to or smaller than the quotient.
Integer Division Practice!

- $5 + 1 \div 2 = 5$
- $9 \div 3 = 3$
- $8 \div 3 = 2$
- $-8 \div 3 = -3$

**Integer division** takes the largest integer that is equal to or smaller than the quotient.
How can I repeat a task a finite number of times?
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1

WARNING: do not use variables on Karel!
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1

This is the same thing as:
    counter = counter + 1
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1

Generally, \( x += y \) is the same as:
\[
    x = x + y
\]
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1

Generally, \( x += y \) is the same as:

\[
x = x + y
\]

You can also do: -=, *=, /=
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1

Computer scientists count from 0.
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1
While loop with variables

counter = 0
while counter < 3:
    do_something()
    counter += 1
For loops
For loop with range

for i in range(3):
    do_something()
For loop with range

```python
for i in range(3):
    do_something()
```

**Definition**

**for loop**

A way to repeat a block of code a specific number of times
For loop with range

```python
for i in range(3):
    do_something()
```

*Tells us we’re going to loop through one by one*
For loop with range

```python
for i in range(3):
    do_something()
```

A variable that helps us keep track of where we are (index)
For loop with range

```python
for i in range(3):
    do_something()
```

Number of iterations
For loop with range

```
for i in range(3):
    do_something()
```

Can be a variable, as long as it’s an int!
For loop with range

```python
for i in range(3):
    do_something()
```

Built-in function
Range

range(3) -> iterates through 0,1,2
Range

range(3) -> iterates through 0,1,2
range(0, 3) -> iterates through 0,1,2
Range

\texttt{range(3)} \rightarrow \text{iterates through 0,1,2}
\texttt{range(0, 3)} \rightarrow \text{iterates through 0,1,2}
\texttt{range(4, 7)} \rightarrow \text{iterates through 4,5,6}
Range

for i in range(end_index):
    # assumes 0 is the start index
Range

```python
for i in range(end_index):
    # assumes 0 is the start index

for i in range(start_index, end_index):
    # end_index is not inclusive!
    # recall: range(4,7) -> 4,5,6
```
How can I make my code more flexible?
Python Functions
def turn_right():
    turn_left()
    turn_left()
    turn_left()
Karel Functions

```python
def move_x_times():
    # ????
```
Karel Functions

def move_x_times():
    # ????

How can we make functions more flexible and reusable by producing different outputs?
Function Analogy

toaster()
Function Analogy

toaster(bread)
Function Analogy

Slide adapted from Chris Piech
Function Analogy

bagel

\texttt{toaster(bagel)}
Function Analogy

You don’t need a different toaster for toasting bagels! Use the same one.
Function Analogy

bagel → toaster(bagel) → toasted bagel
Anatomy of a Function

function(input)
Anatomy of a Function

def function_name(param1, param2):
    result = # do something
    return result
def function_name(param1, param2):
    result = # do something
    return result
Anatomy of a Function

def function_name(param1, param2):
    result = # do something
    return result
Anatomy of a Function

def function_name(param1, param2):
    result = # do something
    return result
def function_name(param1, param2):
    result = # do something
    return result
def function_name(param1, param2):
    result = # do something
    return result

Definition

parameter(s)
One or more variables that a function expects as input
Anatomy of a Function

def function_name(param1, param2):
    result = # do something
    return result

output expected
def function_name(param1, param2):
    result = # do something
    return result

return value
Think/Pair/Share:

Find the function definition, function name, parameter(s), and return value.
Anatomy of a Function

```python
def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
```

Think/Pair/Share:
Find the function definition, function name, parameter(s), and return value in `average`.
def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
def main():  
    mid = average(10.6, 7.2)  
    print(mid)

def average(a, b):
    sum = a + b  
    return sum / 2
Anatomy of a Function

def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
Anatomy of a Function

```python
def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
```

**Definition**

**Return value**
Value that a function hands back to the “calling” function
def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2

Definition

Return value
Value that a function hands back to the “calling” function

What is the “calling” function?
Anatomy of a Function

def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
Anatomy of a Function

def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
Anatomy of a Function

def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2

What’s the difference between arguments and parameters?
Anatomy of a Function

def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2

parameters are the name of input values in the function definition
Anatomy of a Function

def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2

arguments are the values passed in when function is called!
def main():
    mid = average(10.6, 7.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2

Note that we’re storing the returned value in a variable!
Recall from last lecture:

```python
>>> math.sqrt(4)
2.0
```
Recall from last lecture:

```python
>>> math.sqrt(4)
2.0
```
Recall from last lecture:

```python
>>> math.sqrt(4)
2.0
```

*Return value*
Anatomy of a Function

math.sqrt(4)
Think/Pair/Share:

Write a function that takes in two values and outputs the sum of their squares.
Think/Pair/Share:

Write a function that takes in two values and outputs the sum of their squares. [demo]
Functions as Python Objects

def add(x, y):
    return x + y
Parameters and return values are optional

```python
def turn_right():
    turn_left()
    turn_left()
    turn_left()
```

“I’m a function too!”
Parameters and return values are optional

```python
def turn_right():
    turn_left()
    turn_left()
    turn_left()
```

no parameters
Parameters and return values are optional

```python
def turn_right():
    turn_left()
    turn_left()
    turn_left()
    turn_left()
```

no return value
When am I allowed to use a variable?
Scope
Scope Variable Life Expectancy
Definition

**scope**

The parts of a program where you can access a variable
def main():
    function_name()
    print(y)

def function_name():
    x = 2
    y = 3

this is the scope
where x and y “live”
def main():
    function_name()
    print(y)

def function_name():
    x = 2
    y = 3
def main():
    function_name()
    print(y)

def function_name():
    x = 2
    y = 3
Variable Scope

def main():
    function_name()
    print(y)

def function_name():
    x = 2
    y = 3
def main():
    function_name()
    print(y)

def function_name():
    x = 2
    y = 3
Variable Scope

def main():
    function_name()
    print(y)

def function_name():
    x = 2
    y = 3
def main():
    function_name()
    print(y)

def function_name():
    x = 2
    y = 3

NameError
def main():
    function_name()
    print(y)

def function_name():
    x = 2
    y = 3

y is now out of scope!
Variable Scope

```python
def main():
    function_name()
    print(y)

def function_name():
    x = 2
    y = 3
```

Once a function finishes executing, the variables declared inside of it are no longer accessible!
def main():
    y = function_name()
    print(y)

def function_name():
    x = 2
    y = 3
    return y
Unless...

```python
def main():
    y = function_name()
    print(y)

def function_name():
    x = 2
    y = 3
    return y
```

'if we return y, we can use it in main()'
Let’s put it all together!
Receipt program

- What subtasks can we break this program into?
Receipt program

- What subtasks can we break this program into?
  - calculating tax
  - calculating the tip
  - aggregating tax and tip

[demo]
Today’s questions

How do we translate what we know from Karel into regular Python code?

How can we make our code more flexible by producing different outputs depending on the input?
What’s next?
Tomorrow: making programs interactive!

- Strings: representations of text
- Interactive programs