DISCOVERY

#24-26: Control Flow in Python - Loops and Functions

March 15, 2019 and March 29, 2019

Karle Flanagan and Wade Fagen-Ulmschneider

Control Flow in Python, Part II

After writing and exploring many while loops, we find that there is a paradigm that occurs frequently -- **looping a fixed number of times**:

Python:	1
	2
	3
	4
		`i = 0`
		`while i < 5:`
		`  print(i)`
		`  i = i + 1`

Result:		

Python provides a second style of loop that simplifies programming any loops that need to run a fixed number of times called in **for-range-loop**:

Python:	1 2	`for i in range(0, 5):` `  print(i)`
Result:		

Python:	1 2	`for i in range(5):` `  print(i)`
Result:		

Python:	1 2	`for i in range(0, 5, 2):` `  print(i)`
Result:		

The syntax for the range function is defined by Python:

`range( stop ):`
`range( start, stop, [range] ):`

The range type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in for loops.

start: The value of the start parameter (or 0 if the parameter was not supplied)
stop: The value of the stop parameter
step: The value of the step parameter (or 1 if the parameter was not supplied)

-- Python Documentation: https://docs.python.org/3/library/stdtypes.html#range

Puzzle #1:

Python:	`sum = 0`
	`for i in range(10):`
	`  sum = sum + 1`
	`print(sum)`

Result:	

Puzzle #2:

Python:	`sum = 0`
	`for i in range(10):`
	`  sum = sum + i`
	`print(sum)`

Result:	

Puzzle #3:

Python:	`result = 1`
	`for i in range(10):`
	`  result = result * 2`
	`print(sum)`

Result:	

Puzzle #4:

Python:	`result = 1`
	`for i in range(1, 10):`
	`  result = result * i`
	`print(sum)`

Result:	

Puzzle #5:

Python:	`result = 0`
	`for i in range(0, 10, 3):`
	`  if i < 4: result = result + 1`
	`  if i >= 4: result = result - 1`
	`print(result)`

Result:	

Functions

Functions are the final control flow feature we cover in Python and **do nothing until called**. Only when called, the function runs from top-to-bottom and **may return a value back to the caller.** Functions **require** two components and may have two optional components:

1.
2.
3. [Optionally]:
4. [Optionally]:

Puzzle #6:

Python:	1 2	`def rollDie():` `  return random.randint(1, 6)`
Result:		

Puzzle #7:

Python:	1
	2
	3
		`die1 = rollDie()`
		`die2 = rollDie()`
		`print(die1 + die2)`

Result:		

Puzzle #8:

Python:	1 2	`def rollDie(n = 6):` `  return random.randint(1, n)`
Result:		

Puzzle #9:

Python:	1
	2
	3
		`for i in range(1, 4):`
		`  print( rollDie() )`
		`  print( rollDie(i) )`

Result:		

Puzzle #10:

Python:

```
1  def simulate(sides = 6, n = 1000, targetValue = 1):
2    count = 0
3    for i in range(n):
4      roll = rollDie(sides)
5      if roll == targetValue:
6        count = count + 1
7    return count
```

Description:

Function Purpose:

Function Parameters:

sides:

n:

targetValue:

Puzzle #11:

Python:

```
1  simulate( n = 3000 )
2  simulate( 10 )
3  simulate( 20, targetValue = 10 )
4  simulate( sides = 2 )
```

Result:

Puzzle #12:

Python:

```
1  def simulateTaylorsProblem():
2    uniqueBirthdays = 0
3    days = 0
4    while uniqueBirthdays < 365:
5      birthday = random.randint(0, 364)
6      days = days + 1
7      if birthday > uniqueBirthday:
8        uniqueBirthdays = uniqueBirthdays + 1
9
10   return days
```

Description: