Intro to Computer Science

Procedures

A **procedure** takes inputs and produces outputs. It is an abstraction that provides a way to use the same code to operate on different data by passing in that data as its inputs.

Defining a procedure:

```
def <Name>(<Parameters>):
      <Block>
```

The **<*Parameters*>** are the inputs to the procedure. There is one **<*Name*>** for each input in order, separated by commas. There can be any number of parameters (including none).

To produce outputs:

```
return <Expression>, <Expression>, ...
```

There can be any number of expressions following the return (including none, in which case the output of the procedure is the special value **None**).

Using a procedure:

```
<''Procedure''>(<Input>, <Input>, ...)
```

The number of inputs must match the number of parameters. The value of each input is assigned to the value of each parameter name in order, and then the block is evaluated.

If Statements

The **if** statement provides a way to control what code executes based on the result of a test expression.

```
if <TestExpression>:
   <Block>
```

The code in **<*Block*>** only executes if the **<*TestExpression*>** has a **True** value.

Alternate clauses. We can use an **else** clause in an **if** statement to provide code that will run when the **<*TestExpression*>** has a False value.

```
if <TestExpression>:
   <BlockTrue>
else:
       <BlockFalse>
```

Logical Operators

The **and** and **or** operators behave similarly to logical conjunction (and) and disjunction (or). The important property they have which is different from other operators is that the second operand expression is evaluated only when necessary.

- **<*Expression1*>** and **<*Expression2*>**

  - If **<*Expression1*>** has a **False** value, the result is **False** and **<*Expression2*>** is not evaluated (so even if it would produce an error it does not matter). If **<*Expression1*>** has a **True** value, the result of the ***and*** is the value of **<*Expression2*>**.
- **<*Expression1*> or <*Expression2*>**
  - If **<*Expression1*>** has a **True** value, the result is **True** and **<*Expression2*>** is not evaluated (so even if it would produce an error it does not matter). If **<*Expression1*>** has a **False** value, the result of the ***or*** is the value of **<*Expression2*>**.

Loops

Loops provide a way to evaluate the same block of code an arbitrary number of times.

While Loops

A **while** loop provides a way to keep executing a block of code as long as a test expression is **True**.

```
while <TestExpression>:
      <Block>
```

If the **<*TestExpression*>** evaluates to **False**, the **while** loop is done and execution continues with the following statement. If the **<*TestExpression*>** evaluates to **True**, the **<*Block*>** is executed. Then, the loop repeats, returning to the **<*TestExpression*>** and continuing to evaluate the **<*Block*>** as long as the **<*TestExpression*>** is **True**.

Converting between Numbers and Strings

str: str(<*Number*>) = <*String*>

- Outputs a string that represents the input number. For example, **str(23)** outputs the string **'23'**.

ord: ord(<*One-Character String*>) = <*Number*>

- Outputs the number corresponding to the input string.

chr: chr(<*Number*>) = <*One-Character String*>

- Outputs the one-character string corresponding to the number input. This function is the inverse of **ord**: **chr(ord(a)) = a** for any one-character string **a**.

Splitting Strings

split: <*String*>.split() = [<*String*>, <*String*>, ...]

- Outputs a list of strings that are (roughly) the words contained in the input string. The words are determined by whitespace (either spaces, tabs, or newlines) in the string. (We did not cover this in class, but **split** can also be used with an optional input that is a list of the separator characters, and a second optional input that controls the maximum number of elements in the output list.)

Looping through Strings

A **for** loop provides a way to execute a block once for each character in a string (just like looping through the elements of a list):

```
for <''Name''> in <''String''>:
    <''Block''>
```

The loop goes through each character of the string in turn, assigning that element to the **<*Name*>** and evaluating the **<*Block*>**.

This page was last edited on 2015/02/01 23:32:51.

POPULAR NANODEGREE PROGRAMS +

STUDENT RESOURCES +

UDACITY +

INQUIRIES +