Control
Announcements
Pure Functions & Non-Pure Functions

**Pure Functions**
*just return values*

- **Argument**
  - `-2` → `abs`
  - `2`

- **Return value**
  - `2`

- **Two Arguments**
  - `2, 100` → `pow`
  - `126765060022829401496703205376`

**Non-Pure Functions**
*have side effects*

- **Argument**
  - `-2` → `print`
  - `None`

- **Returns None!**

*Python displays the output “-2”*

- **A return value is the value of a call expression and can be used as part of a larger expression**

- **A side effect isn't a value; it's anything that happens as a consequence of calling a function**
Example: Print Then Return

Implement a function $h(x)$ that first prints, then returns, the value of $f(x)$.

```
def h(x):
    return print(f(x))
```

```
def h(x):
    print(f(x))
    return f(x)
```

```
def h(x):
    y = f(x)
    print(y)
    return y
```

(A)          (B)          (C)

What's a function $f$ for which implementations (B) and (C) would have different behavior?

```
>>> h(2)
... h(2)
```

(Demo)
Multiple Environments
Life Cycle of a User-Defined Function

**Def statement:**
- **square(x):**
  - **Return expression:** `return mul(x, x)`

**Call expression:**
- **square(2+2)**
  - **Operator & operands evaluated:**
    - Operator: square
    - Function: `func square(x)`
  - **Function (value of operator) called on arguments (values of operands):**
    - Argument: 4

**Calling/Applying:**
- **Argument:** 4
  - **Signature:** `square(x)`
  - **Body is executed in that new environment:**
    - Return value: 16

**What happens?**
- A new function is created!
- Name bound to that function in the current frame
- Operator & operands evaluated
- Function (value of operator) called on arguments (values of operands)
- A new frame is created!
- Parameters bound to arguments
- Body is executed in that new environment
Control
Conditional Statements

Conditional statements (often called "If" Statements) contain statements that may or may not be evaluated.

| if \( x > 2 \): | print('big') | Two separate (unrelated) conditional statements | big | positive | positive |
| if \( x > 0 \): | print('positive') | | | | |

| if \( x > 2 \): | print('big') | One statement with two clauses: if and elif | big | positive |
| elif \( x > 0 \): | print('positive') | Only one body can ever be executed | | |

| if \( x > 2 \): | print('big') | One statement with three clauses: if, elif, else | big | positive | not pos |
| elif \( x > 0 \): | print('positive') | Only one body can ever be executed | | |
| else: | print('not pos') | | | | |
While Statements

While statements contain statements that are repeated as long as some condition is true.

Important considerations:

• How many separate names are needed and what do they mean?

• The while condition must eventually become a false value for the statement to end (unless there is a return statement inside the while body).

• Once the while condition is evaluated, the entire body is executed.

```
1 i, total = 0, 0  # Names and their initial values
2 while i < 3:     # The while condition is evaluated before each iteration
    i = i + 1       # A name that appears in the while condition is changing
    total = total + i
```

Executed even when is set to 3
Example: Nice Numbers
Nice Numbers

Rounding off 2,799 to 2,800 makes it nice.

**Definition:** A nice number doesn't have 98 or 99 or 01 or 02 among its digits.

Not-so-nice numbers: 99 2,799 5,016 9,902 1,200,456 98,402,001

Nicer versions: 100 2,800 5,000 10,000 1,200,000 100,000,000

These numbers are nice enough already and unaffected: 755 2,859 45,622,895

Implement `nice`, which takes a positive integer `n`. It returns the nearest nice number to `n`.

- For numbers that end in 98 or 99 or 01 or 02, round to the nearest 100.
- Look for 98 or 99 or 01 or 02 among the digits that aren't at the end.

To solve a problem, describe a process and work through an example:

```
4 7 9 8 4 0 2 0 0 1
  ▼
4 7 9 8 4 0 2 0 0 0
  ▼
4 7 9 8 4 0 0 0 0 0
  ▼
4 8 0 0 0 0 0 0 0 0
```
Example: Prime Factorization
Prime Factorization

Each positive integer $n$ has a set of prime factors: primes whose product is $n$

... 
8 $= 2 \times 2 \times 2$
9 $= 3 \times 3$
10 $= 2 \times 5$
11 $= 11$
12 $= 2 \times 2 \times 3$
...

One approach: Find the smallest prime factor of $n$, then divide by it

$$858 \quad = \quad 2 \times 429 \quad = \quad 2 \times 3 \times 143 \quad = \quad 2 \times 3 \times 11 \times 13$$

(Demo)