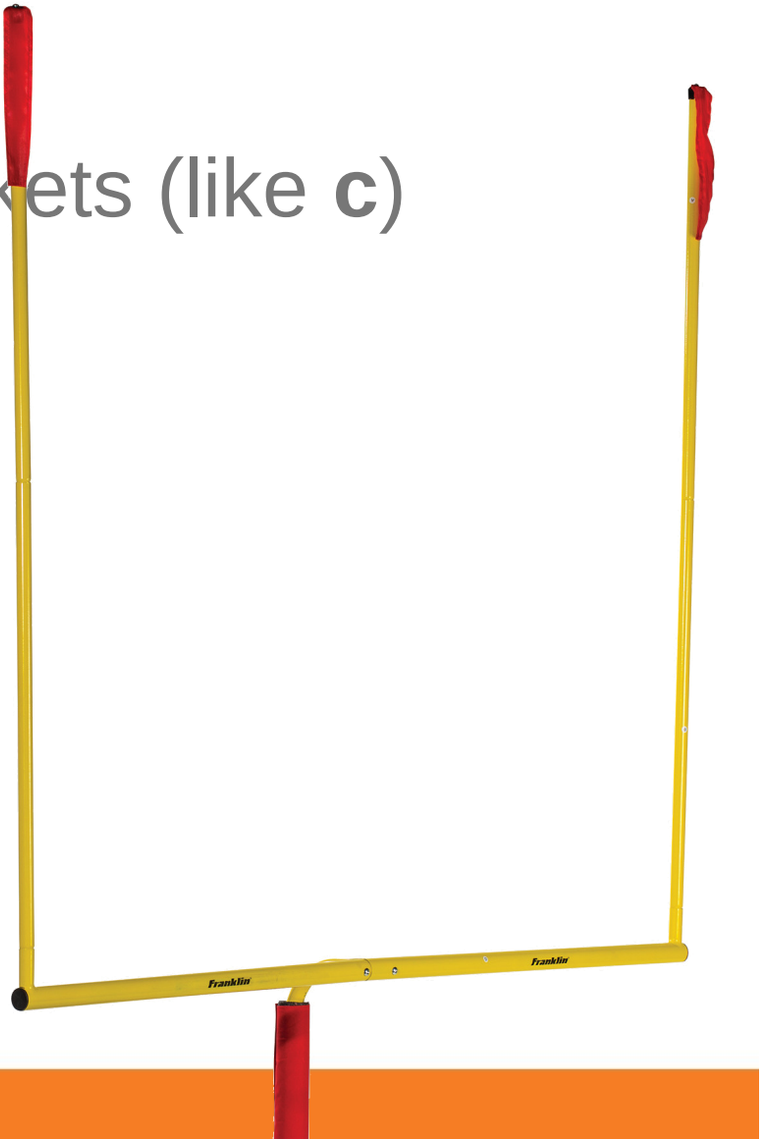
Expressions

Goals

1. Descuss the `pass` keyword
2. Explain why Python doesn't need curly brackets (like **c**)
3. Name 2 loop constructs



Roadmap

1. Syntax Basics
2. Making Decisions
3. Iteration

Develop
Intelligence

Syntax Basics

Develop Intelligence

Variables

- Associate names with values
- Pop into existence on assignment
- Naming rules:
    1. Only letters, numbers, and underscores
    2. Don't start with a number

```
user_id = 12
label = 'Mailing Address'
xs = [0,118,2291]
address_1 = '1234 American Way'
```

Calling Functions

- Invoke with parenthesis (always)
- Pass arguments (sometimes)
- Get something back (sometimes)

```
1 print()
2 print("Hello World!")
3
4 high_scores = [11,23,4,7]
5 result = sum(high_scores)
```

Defining Functions

- Define with `def`

```
1 def print_lyrics():
2     print("I'm a lumberjack, and I'm okay.")
3     print("I sleep all night and I work all day.")
```

- Return something (optional)

```
1 def add(a,b):
2     return a + b
```

Indentation matters

- Lots of languages use braces to denote blocks:

```
function add(a,b){
    return a+b;
}
```

- Python uses colon + indentation

```
def add(a,b):
    return a+b
```

- Used for: functions, loops, conditionals, classes, exception handling

Empty Functions

Develop Intelligence

- The parser is confused by empty functions
  - Or classes or branches
- Use `pass` as a placeholder

```
# Empty function
def do_nothing():
  pass

# Placeholder for when I figure out what to do
if 1==1:
  pass
```

Useful Built-in Functions

Function	What it does	Example
`len`	Gets a sequence length	len("chickens")
`help`	Shows help	help(math)
`str`	Converts to a string	str(datetime.now())
`input`	Gets user feedback	input('[yes/no]')

Line Comments

- Used for annotation
- Don't affect any behavior
- Demarcated with #
- Use judiciously

```
# Below is the id of the user
user_id = 12

# Check if enfarculatable
if is_enfarculatable():
  # Enfarculate!
  enfarculate()
```

Docstring Comments

- String literal
- First statement in a module, function, or class
- Queryable at runtime via `__doc__`
- Good IDEs make it easy

```
def get_sphere_volume(radius):
  """Figures out the volume of a sphere with the
  specified radius.
  """
  pi = 3.14
  return 4*3*pi*(radius**3)
```

Develop
Intelligence

Making Decisions

Develop Intelligence

Good old if

- Keywords `True` and `False`
- Does coersion for non-booleans

```
1  # Boolean literal
2  if True:
3    print('True is true')
4
5  # Comparison operator
6  if 5 > 1:
7    print('Math works!')
8
9  # Coersion
10 if 47:
11   print('I was right all along')
```

Adding Branches

- First condition designated by `if`.
- Any additional conditions designated by `elif`.
- Fallthrough designated by `else`.

```
if color == (0,255,0):
  return 'green'
elif color == (255,0,0):
  return 'red'
else:
  return 'unknown'
```

Equality is Structural

Develop Intelligence

- Equality operator does structural equality
- For reference equality, use `is`

```
1 xs = [1,2,3]
2 ys = [1,2,3]
3 print(xs == ys)
4 print(xs is ys)
5 print('\n')
6
7 ys.pop(0)
8 print(xs == ys)
9 print(xs is ys)
```

Develop
Intelligence

Iteration

Develop Intelligence

Loop Constructs

- Python loops come in just 2 flavors:
  - `for`
  - `while`

Using range

- Lots of languages have a `for` loop:

```
for(int i =0;i<10;i++){
    print(i);
}
```

- Python uses the `range` function:

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

Variations on `for`

Develop Intelligence

```
names=['Rob','Mary','David','Jenny','Chris','Imogen']

# Feels like c
for i in range(len(names)):
    print(names[i])

# More pythonic
for name in names:
    print(name)
```

About `range`

- Three arguments
  - Exclusive max
  - Inclusive min (defaults to 0)
  - Step (defaults to 1)

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

print('\nEquivalent')
for i in range(0,10,1):
    print(i)
```

Fancy range

```
print('Evens')
for i in range(0, 10, 2):
    print(i)

print('\nOdds')
for i in range(1, 10, 2):
    print(i)

print('\nBackwards')
for i in range(9, -1, -1):
    print(i)
```

Keyword `while`

- Executes as long as a condition is true

```
# Sensible use of for + range
for i in range(10):
  print(i)

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

Kicking Out

- Exit the function with `return`
- Exit the loop with `break`

```
i = 0
while True:
  print(i)
  i = i + 1
  if i>=10:
    break
print("done!")
```

Lab: FizzBuzz

Game plan

- In VSCode, open the directory $/python_foundations/labs/
- Read instructions: src/E.O.FizzBuzzTexty/README.md
- Build the applications starting with 'startingpoint'

Develop
Intelligence

Review

1. Descuss the `pass` keyword
2. Explain why Python doesn't need curly brackets (like **c**)
3. Name 2 loop constructs